UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

Detección de Fake News en Twitter usando SNA

Marta Simón Pinacho
Tutor: Álvaro Ortigosa Juárez

Julio 2020

Detección de Fake News en Twitter usando SNA

AUTOR: Marta Simón Pinacho
TUTOR: Álvaro Ortigosa Juárez

Escuela Politécnica Superior
Universidad Autónoma de Madrid
Julio de 2020

Resumen

Este trabajo de fin de grado pretende desarrollar herramientas para caracterizar las relaciones que se establecen entre los usuarios de la red social Twitter de manera que pueda distinguirse cuando la noticia que se transmite entre dichos usuarios es *fake* o no lo es.

En primer lugar, se ha seleccionado un grupo de usuarios de Twitter de los que extraer información tal como algunos de sus seguidores, los tweets publicados más recientemente, los usuarios que los retuitean, etc. Dichas cuentas seleccionadas para la inspección inicial se etiquetan en función de la veracidad que el usuario de la herramienta estime.

A partir de la información recogida, se ha generado una base de datos orientada a grafos en Neo4j sobre la que se han aplicado ciertos algoritmos para obtener características como la centralidad y la popularidad de cada usuario de la red y poder examinarlas más adelante.

A partir de estas características se han seguido dos líneas de análisis:

1. Por un lado, mediante las características calculadas se han entrenado algoritmos de aprendizaje automático como *Random Forest*, Árboles de decisión y *K-Nearest Neighbors* para detectar cuáles de las características definen mejor la red de usuarios que se forma entorno a un tweet falso.

2. Por otro lado, se ha aplicado la técnica de *Graph Embedding* junto con el algoritmo Word2Vec para, mediante la vectorización de las relaciones entre usuarios, ser capaces de etiquetar a un usuario en función de los usuarios con los que tiene una mayor similitud, y a su vez predecir una posible ruta que seguirán los distintos tipos de noticias en función de los usuarios con los que estén relacionados y de su veracidad.

Palabras clave

Redes sociales, Twitter, análisis, Neo4j, Graph Embedding, aprendizaje automático, algoritmo, grafo.

Abstract

This Bachelor Thesis aims to develop tools to characterize the relationships established between users of the Twitter social network so that it can be distinguished when the news that is transmitted between these users is fake or not.

Firstly, a group of Twitter users has been selected from which to extract information such as some of their followers, the most recently published tweets, the users who retweeted them, etc. These accounts selected for the initial inspection are labeled based on the veracity that the user of the tool estimates.

Secondly, based on the information collected, a graph-oriented database has been generated in Neo4j on which certain algorithms have been applied to obtain characteristics such as the centrality and popularity of each user on the network and to be able to examine them later.

From these characteristics, two lines of analysis have been followed:

1. On the one hand, machine learning algorithms such as Random Forest, Decision Trees and K-Nearest Neighbors have been trained using the calculated characteristics to detect which characteristics best define the user network that is formed around a false tweet.

2. On the other hand, the Graph Embedding technique has been applied together with the Word2Vec algorithm to, by vectorizing the relationships between users, be able to tag a user based on the users with whom they have the greatest similarity, and in turn predict a possible route that the different types of news will follow depending on the users with whom they are related to and their veracity.

Keywords

Social network, Twitter, Social network analysis, SNA, Neo4j, Graph Embedding, machine learning, graph algorithms.

Agradecimientos

A mi tutor, Álvaro Ortigosa, por ayudarme y orientarme en las decisiones tomadas durante estos meses de trabajo. Por haber estado pendiente en todo momento de mi progreso, aunque la situación no haya sido la idónea.

A toda mi familia, en especial a mis padres y a mi hermana, por apoyarme siempre en todo lo que hago y estar a mi lado siempre que lo necesito. Gracias por ayudarme a confiar en mí día a día. Sin vosotros, todo esto no habría sido posible.

A mis amigas de siempre, por cuidarme, ayudarme y entenderme. Y a mis amigos de la universidad, con los que he compartido este duro pero bonito camino. Echaré de menos los descansos en la cafetería y las tardes eternas en los laboratorios.

Por último, pero no menos importante, a Andrei, por estar siempre a mi lado, por ayudarme siempre y por haberte convertido en uno de mis pilares fundamentales. Gracias por ser el mejor compañero de vida y por sacar lo mejor de mí cada día.

INDICE DE CONTENIDOS

INDICE DE FIGURAS

INDICE DE TABLAS

1 Introducción

1.1 Motivación

Las *fake news* son un neologismo que se utiliza para referirse a noticias ficticias, es decir, sin base real. Este tipo de noticias, que, a veces, pueden encontrarse en los medios de comunicación tradicionales están más probablemente asociadas a las redes sociales. Se caracterizan por no tener una base real, pero, aun así, se presentan como noticias objetivas, lo que les da una apariencia de verosimilitud.

En la actualidad, las *fake news* están a la orden del día, y más si se habla de redes sociales como Twitter, en las que cualquier persona puede publicar cualquier información desde el más absoluto anonimato. Hoy en día, gracias a Internet, la información puede llegar mucho más lejos y en mucho menos tiempo que en el pasado. No solo esto, sino que, además, se espera que este tráfico de información que se difunde por Internet se triplique durante los próximos años [1].

Este hecho tiene, por supuesto, sus ventajas. Que la información viaje mucho más rápidamente nos da la oportunidad de actuar con mayor rapidez ante las circunstancias que puedan requerirlo. Por el contrario, dicha información, a veces, carece de las suficientes garantías y contraste que toda información que vaya a ser de dominio público debería tener.

1.2 Objetivos

El objetivo principal de este Trabajo de Fin de Grado (TFG) es, mediante el estudio y la utilización de técnicas de análisis de redes sociales o SNA (*Social Network Analysis*), como el cálculo de la centralidad, y el seguimiento del comportamiento de usuarios, aprender a detectar noticias que carezcan de las suficientes garantías de veracidad:

1. Mediante el análisis y el estudio de las características estructurales de las redes de usuarios que se generan alrededor de los distintos tipos de noticias.
2. Buscando patrones de comportamiento en los usuarios.
3. Estableciendo determinadas características que definan a dichos usuarios en función de su tendencia a relacionarse con distintos tipos de noticias.
4. Determinando la forma de propagación de las propias noticias en función de su veracidad.

1.3 Organización de la memoria

La memoria consta de los siguientes capítulos:

- **Estudio del estado del arte.** En este capítulo se introducirán las técnicas y las herramientas que se han utilizado durante el estudio y se pondrá en contexto la situación actual de las *fake news* en las redes sociales
- **Diseño.** En este capítulo se expondrán las decisiones de diseño que se han ido tomando respecto a la utilización de las herramientas explicadas previamente, las características extraídas y clasificadores utilizados para el aprendizaje.
- **Desarrollo.** En este capítulo se explicará con detalle el flujo de trabajo que se ha seguido para desarrollar el estudio. Desde la creación de la base de datos al cálculo de características y la construcción de *datasets* para su posterior clasificación.

- **Integración, pruebas y resultados.** En este apartado se expondrán los resultados obtenidos al aplicar las diferentes técnicas de detección de *fake news* y se hará una comparativa entre ellos.
- **Conclusiones y trabajo futuro.** Por último, a partir de los resultados obtenidos se desarrollarán unas conclusiones y se expondrán posibles mejoras e ideas para trabajos futuros.

2 Estado del arte

2.1 Introducción

En este capítulo se van a introducir los conceptos y herramientas que se han utilizado para la realización de este TFG. En primer lugar, se introduce el término *fake new*; a continuación, se explican las diferentes técnicas de análisis de redes sociales que van a utilizarse más adelante para la detección de *fake news* y se clasifican según sus tipos. Más adelante se introducen las bases de datos orientadas a grafos, en concreto, Neo4j que ha sido la base de datos elegida para llevar a cabo este trabajo. A continuación, se comentan algunos clasificadores y técnicas de aprendizaje automático utilizadas para la detección de patrones. Y, por último, se explica la técnica de *Graph Embedding* empleada para el análisis de las trayectorias de las noticias y la clasificación de los usuarios.

2.2 Las fake news

El término *fake news* es un anglicismo que se ha incorporado al español, apareciendo frecuentemente en el lenguaje común. Se define como un tipo de noticia falsa, en realidad, un bulo formado por un contenido pseudoperiodístico difundido a través de diversos medios y redes sociales, generalmente a través de tecnologías digitales e Internet, y cuyo objetivo es la desinformación [2].

En la actualidad, las *fake news* se han convertido en un problema real puesto que existen ciertos grupos de interés que, sobre la base de noticias ficticias, pretenden influir en hechos del pasado para alterar el curso de acontecimientos futuros. Así, junto con el término de *fake news*, es corriente asociar el término "posverdad". El diccionario de Oxford atribuye el término a Steve Tesich quien lo utilizó por primera vez en 1992. El término se define como relativo o referido a circunstancias en las que los hechos objetivos son menos influyentes en la opinión pública que las emociones y las creencias personales [3].

Las *fake news* están a la orden del día y, debido a esto, cada vez son más las plataformas de difusión de información que cuentan con equipos y algoritmos especiales para su detección. Twitter y Facebook ya han empezado a utilizarlos, permitiendo incorporar etiquetas para indicar si una publicación es falsa o no. Además, están surgiendo diversas plataformas dedicadas específicamente a la detección de *fake news* en la red. Un ejemplo de ello es *maldita.es* [4], una web que se encarga de analizar la mayoría de las noticias virales, contrastarlas y etiquetarlas como bulo en caso de que haya una alta probabilidad de que lo sean. Además de su web, tienen perfiles en redes sociales como Twitter o Facebook en las que publican sus últimas detecciones. Otro ejemplo de este tipo de plataformas es Politifact [5], esta se encarga de detectar *fake news*, pero en vez de etiquetar las noticias como bulos, les asigna un nivel de credibilidad en función de las bases informativas sobre las que se sustente y la calidad del tipo de fuentes de las que proceda. La clasificación que realizan este tipo de plataformas de detección de *fake news* está basada en los conocimientos y la experiencia de las personas que las forman.

Las *fake news* se pueden clasificar de diversas formas. Por ejemplo, según su impacto en la sociedad, la duración del engaño o la emoción que generan, según su interés o el contexto en el que está escrita, según el autor de la noticia, etc. A partir de estas clasificaciones, se pueden realizar dos tipos de análisis para su detección:

- Mediante el procesamiento de lenguaje natural: [6] [7] [8] A partir del análisis del lenguaje natural se puede estudiar el texto, las construcciones sintácticas de las noticias, las emociones que transmiten, etc.
- Mediante el análisis de redes sociales: [9] [10] El análisis de redes sociales permite estudiar la red de usuarios que se genera entorno a una noticia falsa, las relaciones que existen entre tweets (mensajes) y usuarios, etc.

Este segundo análisis será el objetivo principal de este trabajo.

2.3 Análisis de redes sociales

Las redes sociales son uno de los muchos casos de fenómenos de conectividad. Existen muchas otras redes que tienen algunas en características en común como son, por ejemplo, las redes de comunicación, transporte, biológicas, etc. En particular, las redes sociales pueden definirse como un conjunto de usuarios que se relacionan de acuerdo con algunos criterios para intercambiar información.

De acuerdo con un estudio publicado por Statista [11] (empresa líder de mercado en la provisión de estudios de mercado e indicadores estadísticos oficiales), actualmente existen 2,9 billones de personas en el mundo que utilizan las redes sociales en su día a día y se espera que este número siga creciendo en los próximos años.

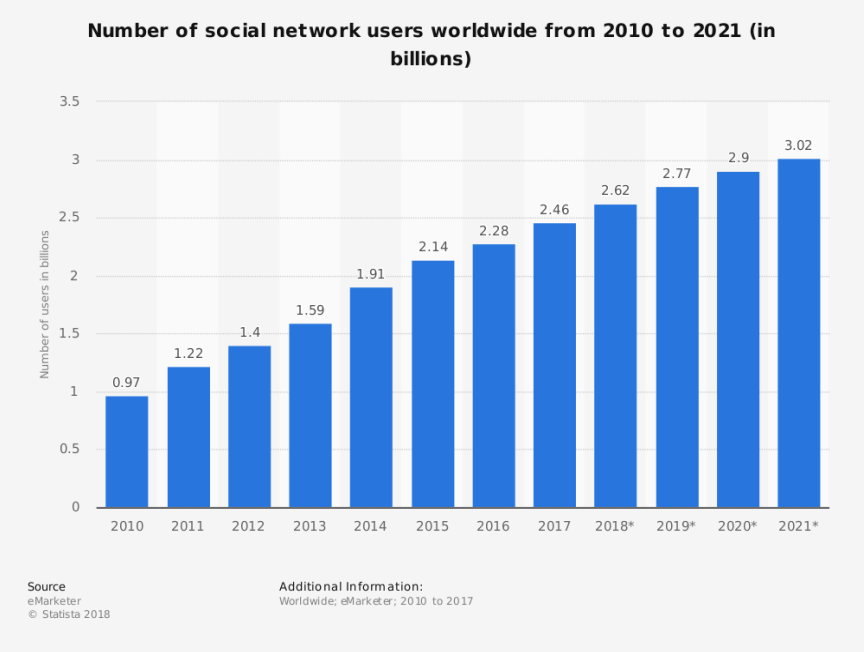


Figura 1: Grafica que muestra el número de usuarios de redes sociales (2010-2021)

En el mundo de las redes sociales existe la idea llamada “seis grados de separación” que contempla que cualquier persona en la tierra puede estar conectada a cualquier otra a través de una cadena de conocidos de una longitud no mayor a seis personas. Esta idea fue establecida originalmente por Frigyes Karinthy en 1929 y desde entonces se ha ido desarrollando hasta llegar a la actualidad, donde se estima que dicha distancia ha disminuido. En la siguiente gráfica se muestra la representación gráfica de esta teoría sobre la red de Facebook. En ella se demuestra que todos los usuarios de la red (un total de 2449 millones) están conectados con cualquiera en una distancia promedio de 3,57 usuarios.

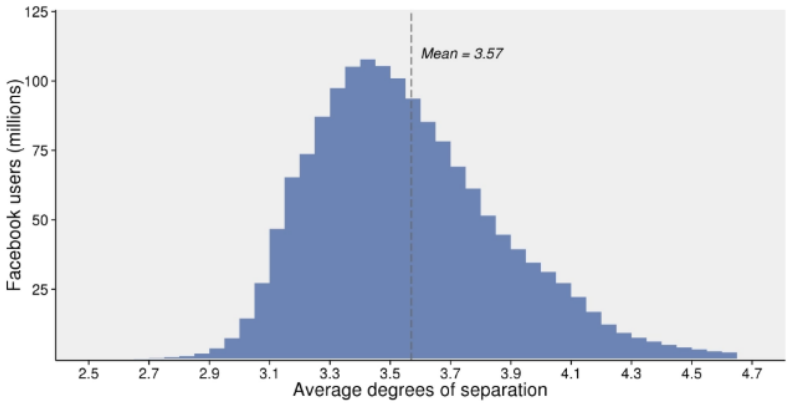


Figura 2: Grafica que representa la media estimada de grados de separación entre todos los usuarios de Facebook (2016)

Conociendo estos niveles de popularidad y el grado de conectividad de las redes sociales, es natural que el estudio y análisis de las redes sociales se haya convertido en una práctica cada vez más arraigada dado que puede ser útil en una gran variedad de ámbitos como la sociología, la ciencia política o, cada vez más, en el área de la industria, el comercio y el turismo ya que los datos que se obtengan de los distintos análisis pueden ser utilizados en decisiones y estrategias de negocio, de marketing y de otros muchos ámbitos.

Dependiendo de los objetivos diseñados, el análisis puede aplicarse de diversas formas. Estos análisis se pueden centrar en distintos elementos de la red, dependiendo de los datos que interese obtener. Pueden analizarse los datos textuales, las relaciones a nivel de usuarios, las acciones y reacciones que realiza cada usuario, etc. [12]

2.3.1 Estudio de la topología y propiedades estructurales

Para el análisis y estudio de las características de las redes sociales, es importante remarcar que la red social se simula mediante un grafo en el que los usuarios son representados en forma de nodo y sus relaciones en forma de arco. Teniendo en cuenta este proceso de adaptación de la red, las métricas con las que se obtienen estas características pueden aplicarse a distintos niveles: [13] [14] [15]

1. Nodos individuales: equivalente a analizar las características de los usuarios de forma individual, por ejemplo, la ubicación del nodo dentro de la red.
2. Arcos individuales: de la misma forma que con los usuarios, el análisis se centra únicamente en las relaciones entre usuarios, como, por ejemplo, número de seguidores (relaciones) que tiene un usuario.
3. El grafo completo: equivalente a analizar las características de la red en su totalidad.

2.3.1.1 Nivel de nodos individuales

A nivel de nodos individuales, podemos calcular diferentes métricas que nos indicarán la popularidad y la importancia de los nodos dentro de la red.

- Cálculo del grado de cada nodo, definido como numero de enlaces entrantes y salientes de cada nodo.
- Cálculo de métricas de centralidad basadas en distancias como, por ejemplo:
    1. El *closeness* que mide el nivel de centralidad de un nodo basándose en su posición relativa al resto de nodos de la red.

2. El *betweenness* que mide la centralidad de un nodo basándose en el número de caminos de distancia mínimo entre dos nodos cualesquiera de la red que tiene que pasar por dicho nodo.
3. El *PageRank*, que es uno de los mejores algoritmos de centralidad conocidos. Consiste en medir la influencia de cada nodo basándose en la influencia de todos sus vecinos.

- El coeficiente de *clustering* de los nodos, que mide la cohesión de un nodo con el resto de la red.

2.3.1.2 Nivel de arcos individuales

A nivel de arcos individuales podemos medir su importancia dentro de la red en función de los usuarios o grupos de usuarios que conecten. La mayoría de los análisis y características que se obtienen de los enlaces tienen que ver con los conceptos de enlaces fuertes o débiles. Algunas de las métricas más populares son:

- El arraigo, que mide el solapamiento de relaciones dentro de la red. Estos arcos son conexiones entre nodos que tienen muchos contactos en común. A su vez esta métrica suele ir relacionada con nociones de fuerza y debilidad de los enlaces.
- La detección de puentes, para encontrar puntos débiles dentro de la red, también está relacionado con el arraigo. Estos puentes pueden ser locales, cuya eliminación implica que la distancia entre los nodos que relaciona aumente, o globales, cuya eliminación implica la creación de una nueva componente conexa independiente del grafo original.
- El *betweenness* que, en el caso de los arcos, se puede utilizar para definir el nivel de conectividad de la red basándose en el número de caminos de distancia mínima que existen entre todos los nodos de la red.

2.3.1.3 Nivel de red global

A nivel de la red global, se pueden aplicar una gran cantidad de métricas. Algunas de ellas han sido explicadas en los dos puntos anteriores, como, por ejemplo, la distribución de grado de la red, similar a la de los nodos, pero extrapolándose a toda la red, o la distribución de los arcos. Al igual que a nivel de nodos, existe una métrica llamada coeficiente de *clustering* a nivel global que mide la cohesión global de la red. Dicha cohesión se define como la probabilidad de que dos nodos de la red con un amigo en común estén unidos directamente.

A nivel de red global, pueden medirse también todo tipo de distancias. Por ejemplo, distancias mínimas promedio entre todos los nodos de la red, diámetro, radio o distancia mínima máxima.

La asortatividad es también una medida que se aplica sobre las redes y que indica en qué medida los usuarios están relacionados con usuarios similares a ellos o distintos. Las similitudes pueden centrarse en el tipo de usuarios, determinadas características que los definan, similitud respecto a la conectividad con el resto de la red, etc. Las redes pueden clasificarse en función de su valor de asortatividad en homofílicas si los usuarios se relacionan con usuarios similares a ellos o heterofílicas en caso contrario.

2.3.2 Modelos de estructura, estudio y detección de comunidades.

Dentro de una red social existen distintas subredes en las que las conexiones son más fuertes. Cuando, además de relaciones más fuertes, los nodos comparten ciertas características en común, dichas subredes se llaman comunidades. Estas comunidades tienen relación con las

redes homofílicas mencionadas antes, debido a que son nodos que están conectados con otros similares a ellos en características.
De las estructuras de las redes también se puede sacar bastante información relevante. Debido a esto existen diversas métricas que pueden ser muy útiles para el proceso:

- La modularidad de la red, se mide por el número de arcos que relaciona nodos del mismo tipo o las mismas características.
- La asortatividad de grado dentro de la red, que mide la preferencia de los nodos de una red por unirse con nodos similares a ellos.

Para resolver el problema de detección de comunidades, pueden aplicarse estas métricas. Se trata de buscar subdivisiones dentro de la red que maximicen la modularidad. Para lograr este efecto existen diversos algoritmos. Algunos de ellos son:

- **Algoritmos basados en bisecciones.** Son algoritmos que parten de dos subdivisiones aleatorias y tratan de trasladar de una subdivisión a otra a la persona o nodo que más haga aumentar la modularidad de ambas. Esta operación se repite hasta que todos los usuarios de la red han sido reubicados. Al final de la ejecución quedan definidas todas las comunidades.
- **Algoritmos basados en uniones**. Su funcionamiento es completamente diferente al de los algoritmos basados en bisecciones, actuando en sentido contrario. Se parte de comunidades formadas por un nodo y se van uniendo dichas comunidades hasta formar una sola. Una vez formada, se retrocede hasta el punto de la ejecución en el que la modularidad fue máxima y así quedan definidas las comunidades.

2.3.3 Modelos de propagación y difusión de información.

Los procesos de difusión de la información por las redes sociales pueden ser de distintos tipos y seguir varios patrones. Con el estudio de estos patrones pueden detectarse comportamientos de la información, rumores, e incluso *fake news*, que es el cometido principal de este proyecto. Dichos procesos de propagación o difusión son, en algunos casos, muy complejos y su estudio podría ayudar a mejorar notablemente la calidad de la información intercambiada en las redes sociales.

Los modelos de propagación y difusión siguen los mismos patrones que los modelos epidemiológicos [16]. Algunos de los modelos que se han definido entorno a la propagación de información son:

- Modelo susceptible – infectado: En este modelo existen dos tipos de usuarios, los que son susceptibles a recibir la información (equivalente a usuarios sanos) y los que ya la han recibido (equivalente a usuarios infectados). En este modelo se define una probabilidad β de contagio o propagación. Una vez que un usuario se contagia no puede curarse. Por lo tanto, este modelo de propagación es exponencial y tiende siempre a 1, donde el que el 100% de los usuarios son contagiados en un tiempo determinado.

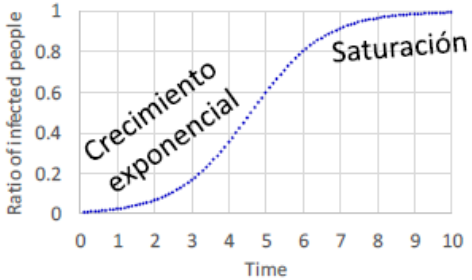


Figura 3: Grafica que indica el crecimiento del modelo de propagación S-I

- Modelo susceptible – infectado – recuperado: Este modelo basado en el modelo anterior, incorpora un factor γ que indica la probabilidad de que un usuario infectado se recupere. Una vez un usuario se ha recuperado se convierte en inmune y ya no puede volver a contagiarse.

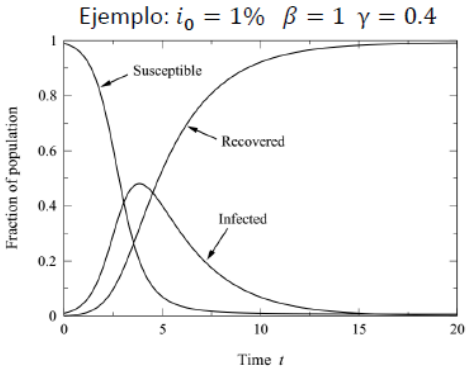


Figura 4: Grafica que indica el crecimiento del modelo de propagación S-I-R

- Modelo susceptible – infectado – susceptible: Se trata de una variante de los modelos anteriores en el cual, una vez que un usuario se recupera no se vuelve inmune, sino que puede volver a contagiarse con una probabilidad que puede variabilizarse.

2.4 Bases de datos orientadas a grafos: Neo4j

Es importante también hablar de las bases de datos orientadas a grafos ya que, como hemos visto en el apartado anterior, la mayoría de los análisis de redes que están relacionados con aspectos estructurales y de relaciones entre usuarios aplican algoritmos sobre grafos. Estas bases de datos pertenecen al grupo de las bases de datos NoSQL, ya que representan su información a modo de nodo y los enlaces entre dicha información a modo de arco.

Además, este tipo de bases de datos tienen diversas ventajas [17], por ejemplo:

- La velocidad de búsqueda depende únicamente del número de relaciones concretas a examinar, no del conjunto de datos global.
- La presentación es intuitiva. Algunos de los gestores de bases de datos, como Neo4j, cuentan con una interfaz gráfica muy amigable que permite analizar de forma gráfica los resultados de las consultas.
- Mantienen estructuras flexibles y agiles que permiten su análisis con rapidez.

También presentan bastante libertad respecto a la asignación de propiedades de los nodos; no es necesario indicar el tipo de la propiedad ni aplicarle ninguna restricción, sino que cada nodo puede contener la información que se considere necesaria en función de lo que represente.

En concreto, para este proyecto se ha utilizado Neo4j, que es la herramienta de gestión de bases de datos orientadas a grafos más popular hasta el momento. Esta herramienta permite representar todo tipo de grafos: dirigidos no dirigidos, con pesos, con etiquetas, etc. Las propiedades que más destacan de Neo4j frente a las bases de datos SQL son:

- Su rendimiento
- Su agilidad a la hora de obtener los resultados de una consulta
- Y la flexibilidad y escalabilidad que tiene, al permitir añadir cualquier tipo de nodos y relaciones a la base de datos.

Además, esta herramienta cuenta con múltiples librerías que son muy útiles a la hora de aplicar SNA sobre grafos, en concreto la librería de ciencia de datos.

2.4.1 Librería Graph Data Science (GDS)

La librería *Graph Data Science* [18] de Neo4j proporciona versiones paralelizadas implementadas de manera eficiente de la mayoría de los algoritmos existentes de análisis de grafos. Todos estos algoritmos se han analizado en el libro *Graph Algorithms* de O'Reilly [19]. La librería GDS permite la aplicación de algoritmos sobre grafos de dos formas: devolviendo los resultados en forma de tabla indicando el valor del resultado de la aplicación del algoritmo sobre los nodos especificados o, alternativamente, escribiendo el resultado del algoritmo directamente en los nodos o en las relaciones del grafo, añadiéndolo como propiedad de estos.

2.5 Detección de patrones mediante clasificadores

Existen numerosos modelos de aprendizaje automático que, a su vez pueden ser entrenados de tres formas diferentes: mediante aprendizaje supervisado, semi supervisado y no supervisado. La principal diferencia entre los tres tipos de entrenamiento es que los modelos con aprendizaje supervisado se entrenan con datos etiquetados, es decir, se conoce el objetivo al que se quiere llegar entrenando la red y se puede comprobar con facilidad su efectividad; mientras que los modelos entrenados mediante aprendizaje no supervisado se entrenan con datos no etiquetados, lo cual permite al clasificador obtener sus propios objetivos en función de los datos de entrada. A la hora de clasificar noticias en función de su veracidad, el objetivo está bastante claro: detectar cuales de ellas son falsas en función de sus características. Por este motivo, en este estudio los clasificadores se entrenan con datos etiquetados.

El principal motivo por el que únicamente se ha estudiado la detección de las *fake news* mediante modelos de aprendizaje automático es debido a que se considera que, adentrarse en el mundo de las redes neuronales quedaba fuera de los objetivos de este TFG. Sin embargo, otra línea de estudio paralela a esta y que podría arrojar muy buenos resultados sería utilizar redes neuronales, como, por ejemplo, las convolucionales, para tratar de detectar dichas noticias.

Algunos modelos de aprendizaje automático han sido estudiados como clasificadores de *fake news*. Entre estos modelos, los más populares son:

- Árboles de decisión
- *Random Forest*
- *K-Nearest Neighbors*

A continuación, se desarrollan brevemente las características de los modelos anteriores.

2.5.1 Arboles de decisión

Un árbol de decisión es un método supervisado de aprendizaje automático que registra todas las posibles consecuencias lógicas de realizar una serie de acciones. El objetivo principal de este clasificador es aprender a predecir el valor de un dato de entrada en función de decisiones lógicas que toma si el dato cumple o no determinados valores. Este clasificador suele estar expuesto al sobre entrenamiento y puede llegar a aportar resultados erróneos si los datos de entrada no están balanceados y existe una clase dominante [20].

2.5.2 Random Forest

El modelo *Random Forest* está basado en la randomización de los árboles de decisión. Se trata de un cultivo de varios árboles de decisión que dan lugar a un bosque. Cada árbol de decisión del bosque señala una clasificación para el mismo dato de entrada y, de entre todas las clasificaciones que obtengamos definidas en función del número de árboles que componga el bosque, el modelo escoge la decisión que más votos tenga. Cuanto mayor sea el bosque, es decir, cuantos más arboles lo formen, el algoritmo clasificará con mayor precisión [21].

En el proceso de creación de estos árboles, se escogen M atributos al azar, siendo M menor que el número total de atributos del dataset de entrada y variándolo durante todo el proceso de creación. Debido a este proceso de creación, en el caso de este modelo de clasificación, no es imprescindible que los datos de entrada estén balanceados, puesto que no llegará a influir en el resultado de la clasificación final.

Además, este modelo elimina el inconveniente del sobre entrenamiento que tienen los árboles de decisión y a su vez elimina su ruido, puesto que es capaz de mantener su precisión en la clasificación incluso cuando faltan una gran cantidad de datos en el conjunto de entrenamiento.

2.5.3 K-Nearest Neighbors

KNN es un método de aprendizaje automático que busca las observaciones más cercanas a la que está tratando de predecir y clasifica el punto de interés basado en la mayoría de los datos que le rodean. Este algoritmo pertenece al dominio de aprendizaje supervisado y puede ser utilizado para el reconocimiento de patrones, extracción de datos y detección de intrusos.

Es un método de clasificación no paramétrico que estima el valor de la función de densidad de probabilidad o directamente la probabilidad a priori de que un elemento cualquiera pertenezca a una clase u otra en función de la información que le proporcionan los datos más cercanos a él. El algoritmo no hace ninguna suposición acerca de la distribución de las variables y es insensible a los valores atípicos [22].

Para este proyecto se ha utilizado la implementación Árboles de decisión [23], Random Forest [24] y KNN [25] de la librería Scikit-learn de Python.

2.6 Graph Embedding

El término *graph embedding* se utiliza para referirse a la transformación de las propiedades de los grafos en un vector o conjunto de vectores. Esta transformación debe capturar la topología del grafo, las relaciones que existen entre los nodos y cualquier otra información que sea relevante para el análisis del grafo, subgrafos o los nodos de estos [26] [27]. Más adelante, estos vectores pueden ser utilizados por algoritmos de aprendizaje automático que toman vectores como entrada para predicción, cálculo de distancia o agrupamiento de los nodos. Un ejemplo de esto sería la clasificación de los nodos, que se puede realizar mediante los vectores de los nodos generados y métodos de aprendizaje automático como KNN o *Random Forest*.

A continuación, se detallan algunas razones por las que es útil realizar la transformación de grafos a vectores:

1. **Las técnicas de aprendizaje automático aplicadas a grafos son bastante limitadas.** Teniendo en cuenta que la gran mayoría de las técnicas actuales de

aprendizaje automático toman como entrada un vector o un conjunto de vectores, el hecho de realizar la transformación del grafo a forma vectorial permite optar a un mayor número de algoritmos para analizar las características de este.

2. **Estos vectores son representaciones comprimidas del grafo**. La matriz de adyacencia de un grafo describe las conexiones que existen entre sus nodos. Se trata de una matriz |N| x |N|, siendo N el número de nodos del grafo, en la cual, cada columna y cada fila representa un nodo. En términos de análisis de redes sociales, en los que se manejan millones de usuarios y relaciones, el uso de la matriz de adyacencia como representación del grafo resulta muy poco eficiente. La utilización de grafos embebidos soluciona este problema ya que se encarga de empaquetar las propiedades de cada nodo en un vector de menor dimensión, lo que hace su análisis mucho más manejable.
3. **Las operaciones de vectores son más rápidas y simples,** en comparación con las operaciones sobre grafos.

2.6.1 Node2Vec

Antes de introducir la técnica Node2Vec utilizada en *graph embedding*, se va a presentar el método Word2Vec en el que se basa dicha técnica. [28]

Word2Vec es una técnica utilizada para aprender *word embedding*, empleada en el procesamiento de lenguaje natural. Se trata de generar vectores a partir de las palabras que componen una listas de oraciones. Los vectores correspondientes a las palabras con significados similares se situarán más cercanos en el espacio. Existen dos algoritmos de entrenamiento para esta técnica:

- **CBOW o *continuous bag of words***. Utiliza el contexto de la palabra para predecir un término objetivo
- **Skip-gram.** En lugar de intentar predecir el término objetivo del contexto, trata de predecir el contexto a partir de un término dado.

Una vez presentada la técnica de Word2Vec, Node2Vec es muy similar [29]. La única diferencia que existe es que, en lugar de utilizar listas de oraciones, se emplean listas de *Random Walks.* Estas rutas aleatorias se obtienen mediante la aplicación del algoritmo *Random Walk [30]*, que pertenece a la familia de los algoritmos encargados de determinar las rutas más cortas. En concreto, este algoritmo se encarga de proporcionar rutas de características aleatorias dentro de la estructura de un grafo.

Una vez obtenidas las rutas aleatorias, se emplea el mismo algoritmo Word2Vec sobre dichas listas para calcular similitudes entre los distintos nodos del grafo, tomando como entrada cada lista de *Random Walk*s en lugar de una frase.

3 Diseño

3.1 Introducción

En este apartado de la memoria se describen las decisiones de diseño que se han llevado a cabo para la realización de este TFG. Se comienza explicando el proceso que se ha llevado a cabo para recolectar la información que se almacena en la base de datos. Continúa definiendo una serie de patrones que serán los que nos ayudarán a detectar similitudes según el tipo de red. Más adelante, se exponen las técnicas de aprendizaje automático y los clasificadores que se utilizarán para la detección de dichos patrones. Y, por último, se explica la aplicación de *graph embedding* como otro posible modo de detección de *fake news.*

3.2 Conjunto de datos.

Para empezar con el desarrollo del trabajo, es necesario contar con una base de datos completa que contenga información sobre tweets y usuarios tanto *fakes* como verídicos. Para ello lo primero que se necesita es tener acceso a la API de Twitter. Existen diversas librerías en Python que nos permiten interactuar con la API. Para este estudio se ha estado utilizando Tweepy [31].

El orden que se ha seguido para la extracción de la información se muestra a continuación.

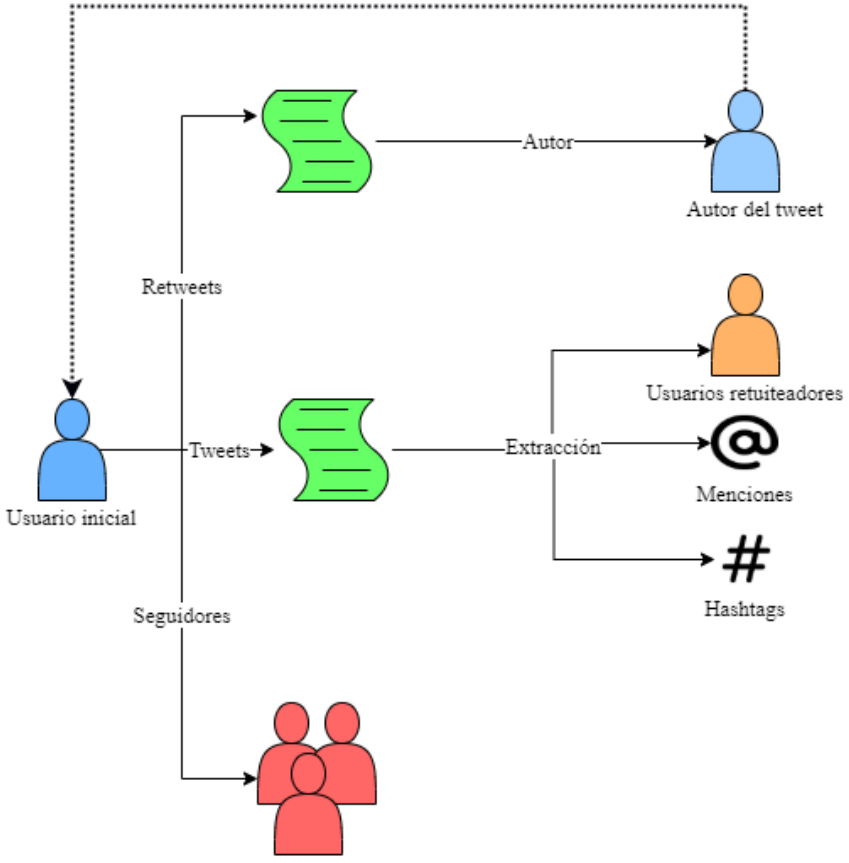


Figura 5:Diagrama que muestra la extracción de la información necesaria para la base de datos

Para comenzar a obtener información de Twitter, se realizó un estudio previo con el fin de obtener un conjunto de usuarios iniciales de los que extraer información relevante para su

análisis. Para la selección de estos usuarios se hizo una recopilación de los usuarios más influyentes en Twitter relacionados con la difusión de noticias y se obtuvo un total de 20 perfiles. A partir de este listado se fue investigando cuales de ellos publicaban un mayor número de veces al día, el número de seguidores que tenía cada uno de ellos, el alcance que tenían las noticias que publicaban según el número de retweets, etc. Una vez hecho esto, se fijó un máximo de cuatro cuentas iniciales a examinar, para que el volumen de datos obtenidos fuese manejable, se seleccionaron las cuentas que se ajustaron mejor a lo que se buscaba y se comenzó a obtener información de ellas.

Las cuatro cuentas seleccionadas fueron *newtral* y *el_pais* como perfiles *no-fakes* (NF), y *malditobulo* y *elmundotoday* como perfiles que distribuyen noticias *fakes* (F). Estos cuatro perfiles fueron etiquetados según su veracidad, 1 para los *fakes* y 0 para los no *fakes*. Así, a la hora de obtener los tweets de los usuarios, se propaga dicha etiqueta para conocer a cuál de las dos subredes de usuarios pertenece cada tipo de tweet. Los usuarios iniciales sobre los que se ha construido la base de datos utilizada en el estudio han sido elegidos con un determinado criterio, y etiquetados en base a dicho criterio, por lo tanto, se puede definir como una base de datos sesgada respecto al criterio elegido.

Comenzó a obtenerse cierta información a partir de estas cuentas. Se optó por obtener un total de 20 seguidores y 10 tweets por cada cuenta inicial. De cada tweet se obtienen los usuarios retuiteadores, las menciones y los hashtags que contengan. En caso de que el tweet obtenido sea un retweet, se procede a añadir al autor original de dicho tweet a la lista de usuarios a inspeccionar, hasta llegar a un total de 10 usuarios inspeccionados. Una vez obtenidos todos los usuarios relacionados tanto con los usuarios iniciales como con los tweets obtenidos se proceden a buscar relaciones de amistad entre dichos usuarios, llegando así a formar una red fuertemente conexa con la siguiente estructura:

Figura 6:Estructura de la base de datos

Actualmente la red generada está compuesta por 1100 nodos y 4000 enlaces.

3.3 Definición de patrones

Como se ha definido anteriormente, con este trabajo se pretende detectar patrones de comportamiento y dispersión de noticias y usuarios sobre la red de Twitter. El estudio de estos patrones incluye la propagación de las noticias por la red, los usuarios que se encargan de propagarlas y las relaciones que existen entre ambos, noticias y usuarios.

La detección de patrones y el cálculo de las características de los nodos de la red se ha aplicado sobre dos subgrafos generados a partir de la base de datos principal, cada uno de ellos formados por tweets de una sola clase (*fakes* o no *fakes*) y todos los usuarios relacionados con dichos tweets. Es decir, un subgrafo que contiene únicamente *fake news* y otro que contiene únicamente noticias verdaderas. Generando estos dos subgrafos se podrán comprobar las diferencias estructurales y de propagación que existan en una red u otra en función de su veracidad.

A continuación, se van a definir los patrones y las características de los nodos de la red que se han obtenido para la detección de *fake news*. [32]

3.3.1 Patrón 1: Usuarios propagadores. Susceptibilidad.

Para la detección de los usuarios que se encargan de propagar los distintos tipos de noticias se puede optar por calcular la susceptibilidad de los nodos que los representan.

En el caso de los usuarios, su susceptibilidad se define como la proporción de noticias falsas que se encargan de difundir respecto al número total de noticias que difunden [32]. Es una formula bastante sencilla que podría expresarse de la siguiente manera:

$$S(u_i) = \frac{\sum_j B(u_i \in G_{F_j})}{\sum_k B(u_i \in G_{T_k}) + \sum_j B(u_i \in G_{F_j})}$$

Donde $B(u_i \in G_x) = 1$ si $u_i \in G_x$ y 0 en caso contrario. En la fórmula, se representa al usuario inspeccionado mediante u_i. G_F y G_T representan los grafos de noticias falsas y verdaderas respectivamente.

Según este cálculo, el valor máximo que puede alcanzar la susceptibilidad de un usuarios es uno, lo que implicaría que el usuario únicamente estuviera relacionado con noticias falsas y, por lo tanto, sería el único tipo de noticia que pudiese difundir.

En el caso de los tweets, también se puede calcular su popularidad. En su caso se puede definir como el número de veces que el tweet ha sido compartido, ya sea mediante retweets o mediante menciones.

Una vez calculada la susceptibilidad se añade como característica de los nodos para su posterior análisis.

3.3.2 Patrón 2: Centralidad de los usuarios.

A la hora de detectar los usuarios con mayor influencia dentro de cada red, se puede acudir a diversas métricas de centralidad. En el caso particular de este estudio, se ha calculado el *closeness*, el *betweenness*, el grado y el *Page Rank* de cada usuario perteneciente a la subred de noticias falsas y a la de noticias verdaderas de forma individual. Todas estas métricas se han obtenido mediante los algoritmos definidos en la librería GDS de Neo4j mencionada en el Capítulo 2 de esta memoria. Además, todas las métricas calculadas se añaden como características de cada nodo.

Una vez obtenidas todas estas métricas, se procede a calcular la media y la mediana para cada nodo, y se incluyen ambos valores como nuevas características de los nodos.

3.3.3 Patrón 3: Diámetro de la red.

Otra característica que interesa conocer, aplicada de forma global a ambas subredes es el alcance de las noticias, que puede calcularse obteniendo el diámetro de la red. Para calcular dicho alcance, se hace uso de la distancia geodésica entre todos los usuarios que pertenecen a cada subred. Esta distancia se obtiene calculando todos los caminos de distancia mínima que existen entre todos los nodos de cada grafo. Habiendo calculado esta propiedad se puede conocer el diámetro de cada subgrafo y compararlos para ver cuál de ellos logra un mayor alcance.

3.4 Generación de datasets

A partir de los valores obtenidos para cada nodo de la red mediante los distintos patrones definidos, se van a generar distintos datasets, modificando el número de atributos y combinándolos de diversas formas hasta llegar a encontrar la combinación que funcione mejor para la detección de noticias falsas.

Para la generación de estos datasets, se obtiene toda la información de todos los nodos de la red y se combinan y almacenan las distintas características. En este estudio hemos generado los siguientes conjuntos de datos:

- **Pattern1.csv**. Contiene el valor de la susceptibilidad calculada para cada usuario y una etiqueta indicando si dicho usuario pertenece a la red de noticias falsas o a la de verdaderas.
- **Pattern2.csv**. Este dataset contiene los valores de todas las centralidades calculadas para todos los usuarios de la red: *betweennes*, *closeness*, *PageRank*, *degree*, media y mediana. Al igual que en el dataset anterior, cada usuario está etiquetado en función de su pertenencia a la red de noticias falsas o a la de verdaderas.
- **Pattern_comb.csv**. Por último, se ha generado un dataset incluyendo todas las medidas obtenidas para todos los usuarios de la red añadiendo la etiqueta correspondiente a la red a la que pertenecen, como ya se ha descrito anteriormente.

3.5 Clasificadores escogidos para el estudio: hiperparámetros.

Una vez obtenidas todas las características que hemos definido en los patrones, pueden ser analizadas mediante modelos de aprendizaje automático para conocer cuáles de ellas son más relevantes a la hora de detectar noticias falsas.

Para este estudio se han escogido tres tipos de clasificadores: arboles de decisión, *Random Forest* y KNN. Una vez obtenidos los resultados se ha comparado qué tipo de clasificador funciona mejor con los distintos tipos de atributos obtenidos.

A continuación, se explican los hiperparámetros de los clasificadores utilizados. En el capítulo siguiente se especificará su valor en función del dataset que se esté utilizando.

3.5.1 Arboles de decisión

Los árboles de decisión de la librería *Scikit-Learn* no están regularizados por defecto, esto quiere decir que, para llegar a un porcentaje de acierto suficiente, se pueden llegar a crear un numero de nodos demasiado elevado que haga que funcione muy bien con los datos de entrenamiento pero que no consiga clasificar nuevos datos, es decir, un sobreajuste en el entrenamiento.

Mediante la regularización de los árboles, se pretende limitar el número máximo de nodos que el árbol puede crear en el entrenamiento y de esta forma, se genere un árbol más simple y que generalice mejor. Algunos de los parámetros a definir para este modelo [23] son los siguientes:

- *Min_samples_split*: especifica el número mínimo de muestras a las que debe de llegar un nodo para dividir la rama y definir dos nuevas decisiones.
- *Max_depth*: indica la profundidad máxima a la que puede llegar el árbol.
- *Min_samples_leaf:* indica el número mínimo de muestras que debe abarcar un nodo final u hoja.
- *Max_leaf_nodes:* especifica el número máximo de nodos hoja que debe tener el árbol

Mediante estas limitaciones se puede llegar a una generalidad que permita clasificar correctamente la mayoría de los datos.

3.5.2 Random Forest

Aunque este modelo funcione relativamente bien aun sin ajustar los hiperparámetros, para conseguir que el modelo aporte mejores resultados, hay que tratar de ajustarlos según el dataset que se esté analizando para tratar de generalizar lo máximo posible y a su vez obtener buenos resultados.

Los principales hiperparámetros a ajustar en este modelo [24] se exponen a continuación:

- *N_estimators:* especifica el número de árboles que se generarán durante el entrenamiento del modelo.
- *Bootstrap:* es un parámetro booleano que indica si se quiere utilizar la muestra de datos completa para entrenar el clasificador o, por el contrario, utilizar diversos tamaños de muestras
- *N_jobs:* se utiliza para paralelizar la ejecución y utilizar diversos núcleos de la CPU para agilizar el proceso de creación de los arboles
- *Oob_Score:* es un método que permite mejorar la precisión y eliminar el sobre entrenamiento mediante la emulación de la validación cruzada.
- *Max_features:* indica el número máximo de características o atributos que compondrán cada árbol.
- *Min_sample_leaf:* indica el número mínimo de muestras que debe abarcar un nodo final u hoja.

3.5.3 K-Nearest Neighbors

Para este clasificador [25] son menos los hiperparámetros que han de definirse para llegar a obtener buenos resultados.

- *N_neighbors*: que define el número de usuarios que se van a utilizar para clasificar cada dato.
- *Weights*: que indica la función que se utilizará durante las predicciones. Las dos posibles funciones son: uniforme, que toma todos los usuarios por igual sin asignar pesos; y *distance*, que asigna un peso como el valor inverso de la distancia entre los dos nodos, de esta forma, los vecinos más próximos al nodo examinado tendrán mayor influencia sobre él.

3.6 Aplicación de Graph Embedding

Otra posible forma de analizar a los usuarios para poder estudiar y detectar cuales de ellos van a ser más propensos a propagar noticias falsas es mediante el estudio de su similitud con el resto de los usuarios de la red. Para el cálculo de estas similitudes se puede utilizar *graph embedding* a nivel de nodo.

Esta técnica es un modo de aplicación de la técnica de *graph embedding* explicada en el capítulo 2 de la memoria. Consiste analizar el grafo a nivel de nodo en lugar de a nivel global, obteniendo así un vector por cada nodo en lugar de un único vector que represente al grafo. Una forma de hacer esto es mediante *random walks* obtenidas a partir de todos los nodos del grafo.

Esta parte del estudio se centrará en la aplicación del algoritmo Node2Vec de la librería *Gensim* [33] a un conjunto de listas de *random walks* obtenidas aplicando dicho algoritmo implementado en la librería GDS de Neo4j sobre el grafo global.

Por lo tanto, para comenzar con el *graph embedding*, lo primero que se necesita es una lista de *random walks.* Para obtenerlas, aplicamos el algoritmo RandomWalk [34] de la librería GDS. Esta implementación del algoritmo tiene algunos parámetros importantes como:

- *Start*: que define el nodo desde el que se comienza la ruta aleatoria
- *Steps*: que define como de larga va a ser la ruta que se va a realizar.
- *Walks*: que define el número de iteraciones que se van a realizar. Al ser rutas aleatorias, cada iteración la ruta será distinta, aunque el nodo origen sea el mismo.
- *Mode*: este parámetro especifica la estrategia que seguirá el algoritmo para escoger la siguiente relación. Puede ser de tipo *random* o Node2Vec. Indicando que el modo de ejecución va a ser Node2Vec, se deben definir otros dos parámetros para terminar de controlar la ejecución de las rutas aleatorias.
    1. *InOut*: este parámetro permite que la búsqueda diferencie entre parámetros "internos" y "externos". Si establecemos este parámetro a un valor mayor que 1, la ruta aleatoria estará sesgada hacia los nodos cercanos al nodo inicial. Dichas caminatas obtendrían una visión local del grafo con respecto al nodo inicial y tendrían un comportamiento similar a BFS (búsqueda en anchura) debido a que las muestras comprenderían nodos de una pequeña localidad. Por otro lado, si establecemos este parámetro a un valor mayor que 1, las caminatas se inclinarán a visitar los nodos más lejanos al nodo inicial, lo que reflejaría un comportamiento similar al de la búsqueda en profundidad (DFS) dentro del marco de la ruta aleatoria.
    2. *Return*: este parámetro controla la posibilidad de volver a visitar inmediatamente un nodo en la ruta. Establecerlo en un valor lo suficientemente alto asegura que sea menos probable que se muestree un nodo ya visitado en las los siguientes dos pasos de la iteración, siempre y cuando el siguiente nodo tenga más vecinos. Esta estrategia fomenta la exploración moderada durante la caminata y evita la redundancia a 2 saltos durante el muestreo. En caso de que se quiera mantener la ruta cercana al nodo inicial, se establecería este parámetro a un valor bajo

Una vez establecido el modo de exploración que seguirán las rutas aleatorias, se obtienen listas de rutas realizadas a partir de todos los usuarios de la red. Dichas listas se introducen como entrada del algoritmo Word2Vec, sobre el que también han de definirse una serie de hiperparámetros como:

- *Window:* establece distancia mínima que debe existir entre la palabra actual y la pronosticada.
- *Min_count:* ignora todos los nodos cuya frecuencia de aparición en las listas de caminos aleatorios sea menor al valor especificado.
- *Sg:* Indica el algoritmo empleado para el entrenamiento. Se establece a 1 en caso de utilizar skip-gram. Con cualquier otro valor se utilizará CBOW.

Una vez entrenado el algoritmo, se pueden obtener las similitudes de todos los usuarios de la red mediante la función *most_similar* de la librería word2vec perteneciente al módulo *gensim.models* de Python. Dicha función calcula la distancia coseno para obtener la similitud entre los usuarios. A partir de las similitudes obtenidas, se puede realizar un proceso de comparación similar al que realiza KNN y clasificar al usuario inspeccionado en función de las características de los N usuarios más similares a él.

4 Desarrollo

4.1 Introducción

En este capítulo de la memoria se explican todos los procesos de desarrollo más técnico que se han llevado a cabo para la realización de este proyecto. Se empieza con una explicación de la aplicación de los algoritmos utilizados sobre el grafo generado. Y, por último, se comentan brevemente las características más relevantes del código desarrollado.

4.2 Aplicación de algoritmos sobre grafos

Para obtener los valores de las características que se han definido en los patrones descritos se han de aplicar los algoritmos correspondientes. Todos los algoritmos que se han usado están definidos en la librería *Graph Data Science* de Neo4j.

Para la aplicación de estos algoritmos se han de seguir unos pasos previos como la estimación de la memoria a utilizar y la creación y gestión de los grafos.

4.2.1 Generación de grafos virtuales

Los algoritmos de grafos se ejecutan sobre una proyección en memoria del grafo Neo4j que reside fuera de la propia base de datos. Esto implica que, antes de ejecutar un algoritmo, ha de crearse una proyección de éste en memoria.
Por lo tanto, un buen hábito es siempre estimar la cantidad de RAM que necesita y configurar un gran tamaño de almacenamiento dinámico antes de ejecutar una carga de trabajo de memoria pesada. A continuación, se muestra el resultado de la estimación de los grafos que se quieren proyectar.

nodeCount	relationshipCount	requiredMemory
1100	3274	"312 KiB"

Figura 7: Estimación de uso de memoria del grafo virtual

Una vez estimado el uso de memoria, se pasa a crear las proyecciones en memoria. En el caso de este estudio, es necesario dividir el conjunto de datos en dos redes distintas: la correspondiente a noticias falsas y la de noticias verdaderas. Para ello se ejecutan las siguientes consultas:

- Creación de la proyección del grafo de noticias falsas:

```
CALL gds. graph.create.cypher("All_fakes",
        "MATCH (n: TwUser)<-[]->
(m:Tweet {is_fake : '1'}) RETURN id(n) as id
Unión MATCH (l: TwUser)-[:FOLLOWS*1..2]->(n:TwUser)<-[]->
(m:Tweet {is_fake : '1'}) return id(l) as id
union MATCH (n: Tweet {is_fake: '1'}) return id(n) as id",
        "MATCH p=(l: TwUser)-[r:FOLLOWS*1..2]->(n:TwUser)<-[]->
(m:Tweet {is_fake : '1'}) UNWIND relationships(p) as rel return id(l) as
 target, id(n) as source, collect(distinct type(rel)) as types")
```

- Creación de la proyección del grafo de noticias verdaderas:

```
CALL gds. graph.create.cypher("All_non_fakes",
        "MATCH (n: TwUser)<-[]->
(m:Tweet {is_fake : '0'}) RETURN id(n) as id
Unión MATCH (l: TwUser)-[:FOLLOWS*1..2]->(n:TwUser)<-[]->
(m:Tweet {is_fake : '1'}) return id(l) as id
union MATCH (n: Tweet {is_fake: '0'}) return id(n) as id",
        "MATCH p=(l: TwUser)-[r:FOLLOWS*1..2]->(n:TwUser)<-[]->
(m:Tweet {is_fake : '0'}) UNWIND relationships(p) as rel return id(l) as
 target, id(n) as source, collect(distinct type(rel)) as types")
```

4.2.2 Cálculo de características y algoritmos aplicados

Una vez proyectados los grafos sobre los que interesa definir los patrones, se procede a aplicar los algoritmos seleccionados sobre ellos y a calcular las características que interesan en el estudio de la estructura.

Todos los algoritmos que se han utilizado han sido aplicados sobre los dos grafos proyectados. Más adelante se compararán los valores de ambos resultados para comprobar si existen o no diferencias estructurales entre ambas redes.

Existen tres modos diferentes de ejecutar algoritmos sobre grafos.

- Modo Write. El modo escritura permite almacenar como propiedad el valor calculado por el algoritmo para cada nodo o relación del grafo, estando disponible puesto que se almacena de forma permanente en el grafo de la base de datos, aunque el algoritmo se aplique sobre la proyección.
- Modo Stats. Sigue la misma línea de ejecución que la modalidad de escritura, pero, al contrario que esta, los valores calculados por el algoritmo se escriben sobre la proyección en memoria del grafo, no sobre la base de datos real, por lo tanto, no permanecen, una vez se elimina la proyección del grafo, se pierden los valores calculados.
- Modo Stream. Esta modalidad de ejecución de los algoritmos devuelve los valores calculados a modo de tabla, indicando el valor calculado para cada nodo. Los algoritmos ejecutados mediante esta modalidad no almacenan los resultados en ningún momento, únicamente los muestran por pantalla.

4.2.2.1 Grado de los nodos.

A la hora de calcular el grado de los nodos, no se ha aplicado ningún algoritmo de la librería GDS. En su lugar, se ha calculado el número de enlaces entrantes y salientes que tiene cada nodo y mediante la suma de ambos se ha obtenido el grado total de cada usuario de la red. Una vez obtenido el valor del grado de cada nodo se ha añadido como propiedad de estos.

A continuación, se muestra una tabla que contiene las primeras filas resultado de la consulta realizada para obtener el grado de cada nodo.

screen_name	degreeOut	degreeIn
IncapacidadP	31	18
Yemuyin	28	18
Carmen000Carmen	28	12
ChristiansCCR	26	13

JuanCar44257185	23	15

Tabla 1: Muestra de los primeros resultados de la consulta para calcular el grado de los nodos del grafo.

4.2.2.2 Betweenness

Para el cálculo de la centralidad mediante el algoritmo *betweenness* se ha utilizado el algoritmo definido en la librería GDS.

Este algoritmo se ha aplicado de forma independiente sobre el grafo de noticias falsas y de verdaderas en modo de escritura, de tal forma que el valor calculado para cada nodo se añade directamente como propiedad de este.

A continuación, se muestra un ejemplo del resultado de la aplicación del algoritmo. En la imagen se muestra el valor que cada nodo ha obtenido tras haberles aplicado el algoritmo. El nodo gris que no contiene información en su interior se corresponde con un nodo del tipo Tweet, sobre los cuales no se están aplicando los algoritmos. En la imagen tampoco se muestran todas las relaciones de todos los nodos, puesto que si no sería difícil de observar.

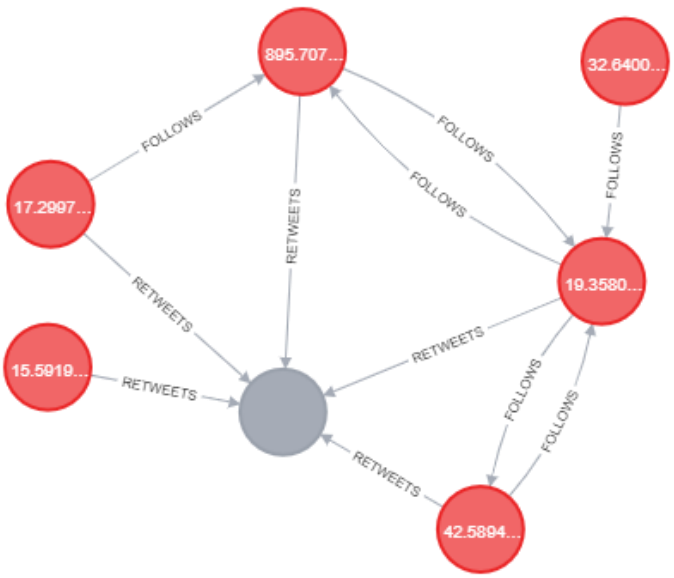


Figura 8: Muestra de los resultados de la ejecución del algoritmo *betweenness* sobre el grafo.

4.2.2.3 Closeness

Al igual que el algoritmo de *betweennes*, utilizamos el algoritmo *closeness* de la librería GDS de Neo4j para la aplicación de este algoritmo en modo escritura para almacenar de forma permanente los resultados obtenidos y así poder analizarlos.

A continuación, se expone otra muestra del resultado de la ejecución, en este caso del algoritmo *closeness* sobre nuestro grafo. Los valores que contienen los nodos corresponden al valor de *closeness* que el algoritmo ha calculado para cada uno de los nodos.

Figura 9: Muestra de los resultados de la ejecución del algoritmo *closeness* sobre el grafo.

4.2.2.4 Page Rank

Por último, se ejecuta el algoritmo *Page Rank*, de nuevo con la modalidad de escritura para almacenar los resultados. En el caso de este algoritmo, es necesario indicar algunos parámetros antes de iniciar la ejecución, en concreto, el número máximo de iteraciones que queremos que realice el algoritmo y el *damping factor*, que simboliza la probabilidad que existe de ir de un nodo a otro. Para esta ejecución en concreto, se han especificado un total de 20 iteraciones máximas y un *damping factor* de 0.85, que suele ser el valor habitual.

A continuación, se muestra una pequeña parte del grafo con los resultados de la aplicación del algoritmo *Page Rank*.

Figura 10: Muestra de los resultados de la ejecución del algoritmo *Page Rank* sobre el grafo.

4.2.2.5 Diámetro de la red

Como se comentó en el Capítulo 2 de esta memoria, el diámetro de la red es la mayor distancia geodésica existente en la red. Por lo tanto, para el cálculo del diámetro de la red, se necesita calcular la distancia geodésica entre todos los nodos de cada una de las dos redes por separado. La máxima distancia que se obtenga será el diámetro de la red.

Para calcular la distancia geodésica, se aplica el algoritmo *allShortestPaths* de la librería GDS de Neo4j. Una vez obtenidas las distancias, se ordenan de mayor a menor, la primera fila que devuelva la consulta será el diámetro de la red.

4.2.2.6 Random Walks

Como se ha explicado en el capítulo de diseño de esta memoria, para aplicar *graph embedding* necesitamos una lista de *random walks* partiendo de todos los usuarios del grafo. Para ello, se ha configurado el algoritmo para que realice un total de 2 iteraciones por ruta, con un máximo de 5 nodos de distancia al inicial. A la hora de definir los parámetros para el algoritmo Node2Vec, se decidió establecer el valor del parámetro *inOut* a 0,4, para realizar una búsqueda en profundidad por el grafo y llegar a abarcar nodos suficientemente lejanos al inicial para obtener una muestra lo más heterogénea posible, y el valor del parámetro *return* a 1 para tratar de evitar la repetición de los nodos durante la exploración de los caminos.

A continuación, se muestra un ejemplo de ejecución del algoritmo *Random Walk*, estableciendo como nodo inicial al usuario “el_pais”. La salida de la ejecución es la siguiente:

Walks
["el_pais","Planeta_Futuro","aramayuelas","1001Medios","Montesjulio","maldita_es"]
["el_pais","elpaiscat","tgarciaramon","czanon","RebecaCarranco","tgarciaramon"]

Tabla 2: Resultados de la aplicación del algoritmo RandomWalk

Se puede observar que el algoritmo devuelve un total de 5 nodos por iteración y realiza un total de 2 iteraciones. Al haber establecido el parámetro *return* a 1, no se repite prácticamente ningún nodo en ninguna de las iteraciones. A continuación, se muestran las relaciones que existen entre los usuarios devueltos por el algoritmo:

Figura 11: Relaciones entre los nodos devueltos por el algoritmo RandomWalk

En la imagen se observan dos caminos bastante diferenciados. Al la izquierda del nodo central, que ha sido el nodo inicial, se sitúan los nodos pertenecientes al camino devuelto en la primera iteración y a la derecha, los de la segunda iteración. En el primer camino existen un total de 5 nodos, algunos de ellos relacionados entre sí. En el segundo camino, hay un total de 4 nodos puesto que hay uno que se repite al estar conectado con todos los demás nodos del camino.

A continuación, se muestran las propiedades de los nodos de la red tras la aplicación de los algoritmos explicados:

- Los nodos de tipo Tweet:
    1. **Id:** Contiene el id del tweet devuelto por la api de Twitter.
    2. **Is_fake:** Indica si el tweet ha sido extraído de uno de los usuarios **iniciales** etiquetados como *fake* (1) o como no *fake* (0).
    3. **Susceptibility:** Contiene la popularidad calculada para los tweets
    4. **Text:** Contiene el texto original del tweet extraído de la api de Twitter
- Los nodos de tipo Usuario:
    1. **Betweenness:** Almacena la propiedad de betweenness calculada
    2. **Closeness:** Almacena la propiedad de closeness calculada
    3. **Degree:** Almacena la suma de grado de entrada y salida del nodo
    4. **PageRank:** Almacena la propiedad de PageRank calculada
    5. **Mean:** Almacena la media calculada sobre las cuatro propiedades de centralidad de los nodos
    6. **Median:** Almacena la mediana calculada sobre las cuatro propiedades de centralidad de los nodos
    7. **Susceptibility:** Contiene la medida de susceptibilidad calculada para los usuarios
    8. **Screen_name:** Contiene el nombre del usuario al que representa el nodo

4.3 Explicación del código generado

Todo el código generado para este trabajo ha sido desarrollado en Python. Se han desarrollado un total de 5 módulos distintos.

4.3.1 Twitter_api.py

Este script se encarga de obtener información de la API de Twitter a través de la librería Tweepy. Al inicializar la clase se establece la conexión con la API mediante las claves de acceso que se han solicitado previamente. Además de la conexión, se han desarrollado métodos para extraer los tweets de los usuarios, las menciones y los hashtags de cada tweet, los seguidores, y los usuarios retuiteadores. Todas las funciones devuelven los datos obtenidos en un formato JSON para introducirlos más adelante a la base de datos.

4.3.2 Data_to_graph.py

Este script se encarga de la conexión con la base de datos. Una vez generada la base de datos, se procede a la comunicación desde Python mediante la librería Py2Neo, desde la cual se obtiene el grafo para poder interactuar con él. En este script se definen todas las funciones relacionadas con la inserción de datos en el grafo: inserción de tweets y de usuarios y relaciones entre ambos. Se definen las restricciones necesarias a la hora de la creación del grafo tales como nombre de usuario, id de los tweets y términos de los hashtags únicos. Y se establecen los índices sobre el texto de los tweets para su posterior procesado.

Para esta base de datos se ha seguido la estructura que ha marcado el tutor, para la posterior integración con su base de datos.

- TwUser: son nodos que representan usuarios, su clave primaria es el nombre de usuario.
- Tweet: son nodos que representan los tweets. Su clave primaria es el id del tweet y en ellos se almacena el texto, y la fecha de creación de los tweets.
- Hashtag: son nodos que representan los hashtags que utilizan los tweets. En ellos se almacena el término que compone el hashtag.
- Follows: representan las relaciones de amistad entre usuarios indicando la dirección de la relación.
- Mentions: representan una relación entre un tweet y un usuario. Indican cuando un tweet contiene en su texto una mención a un usuario.
- Authors: son relaciones entre usuarios y tweets que indican que usuario es el autor de cada tweet.
- Retweets: representan las relaciones entre usuarios que han retuiteado un determinado tweet.
- Uses: es la relación existente entre tweets y hashtags que se crea cuando un tweet utiliza un determinado hashtag en su texto.

4.3.3 Database_generator.py

Una vez generados los dos módulos anteriores, se procede a la unión de ambos en un programa principal que contiene un objeto de la clase Twitter_api para obtener la información y otro de la clase data_to_graph para introducirla dentro de la base de datos una vez obtenida.

4.3.4 Database_analysis.py

Este script se encarga de la aplicación de los algoritmos explicados sobre el grafo. Obtiene datos ejecutando consultas mediante py2neo, los convierte en dataframes de pandas para interactuar con ellos más fácilmente, realiza los cálculos pertinentes y escribe los resultados en el grafo. Las principales funciones que componen este módulo son:

- Cálculo de la centralidad. En la que se aplican los algoritmos de centralidad explicados en el capítulo dos de esta memoria y se escriben los valores obtenidos como características de los nodos. Además, se calcula la media y la mediana de cada nodo y también se añaden como característica de los nodos.
- Cálculo de la susceptibilidad de los usuarios y la popularidad de los tweets.
- Cálculo de las distancias entre vectores que representan a cada usuario.

Además, en este script también se desarrolla todo el proceso de *graph embedding*, desde la ejecución del algoritmo *Random Walk*, hasta el procesado de los resultados y la aplicación de Word2Vec.

4.3.5 Pattern_detection.py

En este módulo se aplican los modelos de aprendizaje automático que se han explicado con anterioridad. Para poder aplicar los clasificadores es necesario obtener el conjunto de datos con el que se quiere entrenar el modelo. Por ello se ha creado una función que obtiene los datos de todos los usuarios de la red y genera distintos ficheros con distinto número de atributos en cada uno para, después, estudiar cual es la mejor combinación de atributos para detectar noticias falsas. En este módulo también se definen todos los clasificadores empleados en el estudio, se entrenan y se clasifican los datos perteneciente al dataset que se quiera analizar y se obtienen las gráficas de las curvas ROC, resultantes de la clasificación de dichos datos.

5 Pruebas y resultados

5.1 Introducción

Una vez se han obtenido las características más relevantes de todos los nodos de la base de datos se procede a analizarlas, compararlas y clasificarlas. Las pruebas que se han realizado para este estudio han sido:

1. Búsqueda de diferencias estructurales entre redes de noticias falsas y verdaderas.
2. Entrenamiento y clasificación de los datos mediante los algoritmos de aprendizaje automático *Random Forest*, Arboles de decisión y KNN.
3. Aplicación de *graph embedding* mediante el algoritmo Node2Vec y estimación de niveles de confianza de las clasificaciones obtenidas.

5.2 Comparación a nivel estructural

La primera característica a nivel estructural que se puede remarcar es el número de nodos perteneciente a cada subred. Para comenzar con la comparación, partiendo de la base de que se ha realizado la misma inspección sobre todos los usuarios iniciales, sean de la clase que sean, en la siguiente gráfica se observa que, en la base de datos generada existe un mayor número de usuarios relacionados con la clase *fake* que con la no *fake*.

Figura 12: Proporción de tweets falsos vs verdaderos

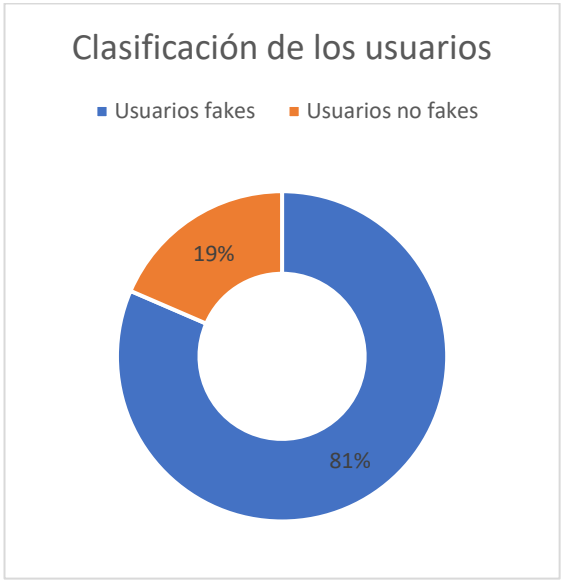


Figura 13: Proporción de usuarios pertenecientes a la red *fake* vs lo pertenecientes a la red no *fake*.

Se puede ver que, en relación con los tweets de la base de datos la diferencia no es demasiado significativa, ya que prácticamente la mitad de los tweets pertenecen a la clase *fake* y la otra mitad a la no *fake*, como era de esperar debido a la recolecta de datos que se realizó. Sin embargo, en relación con los usuarios de la red, hay una diferencia bastante notoria entre el número de usuarios que están relacionados con la red *fake* y los que lo están con la red no *fake*. Esto se debe a que, aunque se haya obtenido la información de la misma forma para todos los usuarios, existen usuarios que pertenecen a ambas subredes, es decir, que están relacionados con tweets *fakes* y no *fakes*. Dichos usuarios tendrán una susceptibilidad menor que los usuarios que únicamente se relacionan con noticias *fakes*, pero, aun así, han sido

catalogados como pertenecientes a la subred *fake* para simplificar la clasificación. Más adelante se observarán las diferencias entre los usuarios con diferente susceptibilidad.

Otra característica estructural para comparar es la distribución de grado de ambas redes. Esta comparación se realiza para observar si existe alguna diferencia entre los tipos de relaciones de los usuarios que pertenecen a la red *fake* y los de la red no *fake*. Para ello, se obtiene el grado de todos los nodos de la red y se representan sus frecuencias. A continuación, se muestran las gráficas de distribución de grado de grafo completo y de ambas subredes por separado.

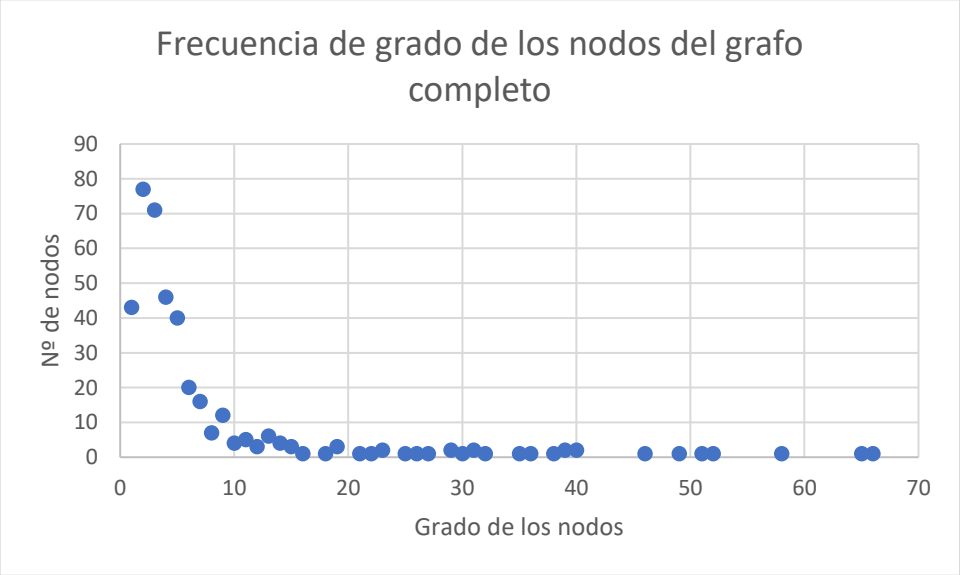


Figura 14: Distribución de grado de los nodos del grafo completo

En esta gráfica podemos observar que la distribución de grado del grafo completo sigue una distribución *Power Law*, es decir, existe una acumulación de nodos en los valores más pequeños, y según aumenta el valor del grado, estos valores van disminuyendo.

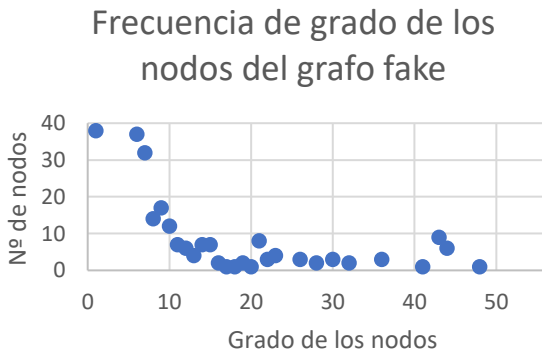


Figura 15: Distribución de grado de los nodos del grafo *fake*

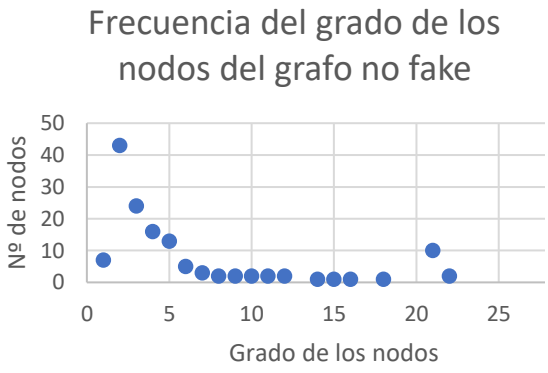


Figura 16: Distribución de grado de los nodos del grafo no *fake*

A la hora de detallar la distribución del grado de los nodos de ambas subredes, se puede ver una diferencia en cuanto al valor del grado en la red *fake*, que cuenta con valores de grado más altos que en la no *fake*. Esto es debido a que los usuarios de las redes *fake* están más conectados entre sí, permitiendo una mayor rapidez en la propagación de las noticias.

A la hora de comparar los diámetros de las subredes, se puede ver que la red *fake* en este caso tiene un diámetro menor que la no *fake*. La principal razón de este hecho es que, al estar menos conectados los nodos de la red no *fake*, y debido a la forma de calcular el diámetro mediante la máxima distancia mínima entre los nodos de cada red, dicha distancia entre dos nodos de la red *fake* va a ser menor, puesto que todos sus nodos tienen un grado mayor y las distancias mínimas serán menores que las de la red no *fake*.

Red global	Red fake	Red no fake
11 nodos	6 nodos	7 nodos

Tabla 3: Resultado de los cálculos de los diámetros de las redes

Con respecto a las medidas de centralidad calculadas para los nodos de las dos subredes, se ha observado que los valores de todas las medidas tomadas sobre los nodos del grafo *fake* son mayores que los del grafo no *fake*. Esto se debe, en parte a la distribución de grado que se ha explicado anteriormente. Los nodos del grafo *fake* tienen un grado mayor que los del no *fake* y esto puede suponer que sus medidas de centralidad sean mayores, como se observa en las gráficas.

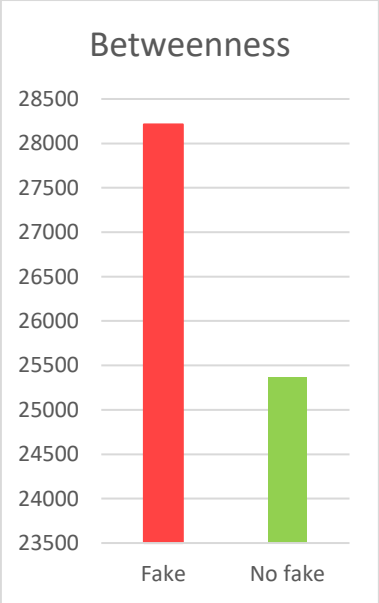


Figura 17: Comparación del valor máximo de *betweenness* en las dos subredes.

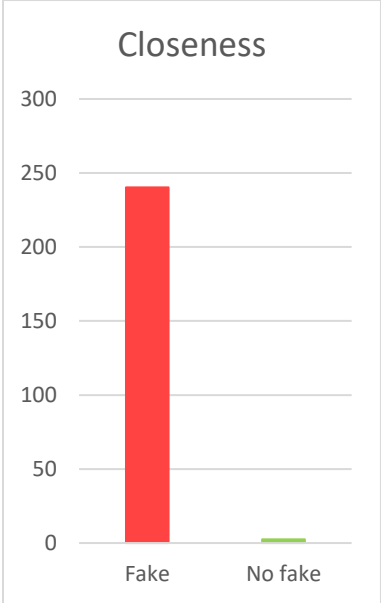


Figura 18: Comparación del valor máximo de *closeness* en las dos subredes.

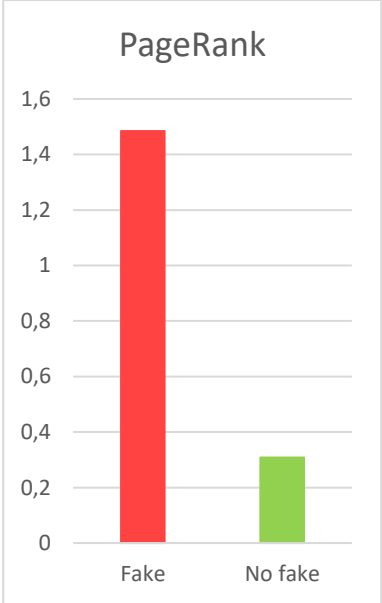


Figura 19: Comparación del valor máximo de P*age Rank* en las dos subredes.

Si se compara el proceso de difusión de las noticias con los modelos de propagación de las redes sociales descritos en el Capítulo 2 de esta memoria, esta difusión se corresponde con un modelo susceptible – infectado (S-I) ya que, las noticias se propagan por la red hasta llegar a "infectar" al mayor número de usuarios posibles.

Dado que, como hemos observado, existe un mayor número de usuarios en la red *fake* que en la no *fake*, y teniendo en cuenta que el parámetro β, que indica la probabilidad de contagio o, en este caso, de propagación de la noticia, tiene el mismo valor para ambas redes, pues la probabilidad de difusión es la misma, podemos determinar que las noticias falsas o *fake*, al

estar relacionadas con un mayor número de usuarios, lograrán un mayor alcance que las verdaderas.

5.3 Utilización de clasificadores para detección de patrones

Para empezar a clasificar los datasets generados con los distintos clasificadores elegidos, lo primero que hay que hacer es dividir los datos en entrenamiento y clasificación. Como se ha visto en el apartado anterior de este mismo capítulo, existe un mayor número de datos referentes a noticias *fakes* que a *no fakes*, es por eso por lo que, a la hora de dividir los datos mediante la función *train_test_split* de Scikit-learn, se utiliza el parámetro *stratify* para asegurar que tanto en los datos de entrenamiento como de clasificación están representadas las dos clases. Se ha decidido utilizar un 70% de los datos para entrenamiento y un 30% para clasificador en todas las pruebas realizadas. Una vez separados los datos, se puede empezar con la aplicación de los clasificadores.

La intención principal de estudiar estos tres datasets con distintas combinaciones de las características de los usuarios, es determinar cuál de las características calculadas es más influyente a la hora de clasificar a un usuario como perteneciente a una red *fake* o *no fake.*

En el Anexo A se exponen muestras de todos los datasets generados, para poder tener una visión global de los datos. En el Anexo B se muestran las curvas ROC y las matrices de confusión resultantes de todas las ejecuciones con los tres clasificadores, para mostrar la sensibilidad de los resultados obtenidos.

5.3.1 Dataset 1: Susceptibilidad.

Este dataset es el más sencillo, puesto que únicamente tiene un atributo (susceptibilidad) y la etiqueta a la que corresponde dicha susceptibilidad F o NF.

Para la clasificación de este dataset, los parámetros utilizados han sido los siguientes:

1. **Random Forest:** Se han definido un máximo de 5 árboles y un total de 1 característica, puesto que es la única que contiene el dataset. Tras el entrenamiento, se ha observado que la media del número de nodos por árbol generado ha sido de 5 y la profundidad máxima de cada árbol ha sido de 2. Además, con estos parámetros se ha obtenido una tasa de acierto en test de 0’99.
2. **Árbol de decisión:** Para este clasificador se ha definido una profundidad máxima de 5 nodos y un máximo de 5 nodos hoja. Tras el entrenamiento, se ha observado que el árbol generado ha tenido un total de 7 nodos y una profundidad total de 3 Con estos parámetros se ha obtenido una tasa de acierto del 0’99 en test.
3. **KNN:** Para este clasificador se han definido un total de 5 vecinos para tener en cuenta. Con esta configuración, el clasificador ha obtenido una tasa de acierto en test de 0’99

En general, se puede ver que con este dataset se han obtenido resultados muy buenos con respecto a la clasificación de los usuarios. Esto indica que el Patrón 1, definido en el apartado 3.3.1 de esta memoria, permite detectar, en función de la susceptibilidad del usuario, el tipo de red a la que pertenece.

5.3.2 Dataset 2: Centralidad.

Este dataset cuenta con un total de seis atributos: b*etweenness*, closeness, *Page Rank*, grado, media y mediana.
Para la clasificación de este dataset, los parámetros utilizados han sido los siguientes:

1. **Random Forest:** Se han definido un máximo de 100 árboles y un total de 5 características, que son todas las que contiene el dataset, es decir que todos los árboles generados tendrán en cuenta todos los atributos del dataset en sus decisiones. Tras el entrenamiento, se ha observado que la media del número de nodos por árbol generado ha sido de 95 y la profundidad máxima de cada árbol ha sido de 16. Además, con estos parámetros se ha obtenido una tasa de acierto en test de 0'95.
2. **Árbol de decisión:** Para este clasificador se ha definido una profundidad máxima de 20 nodos y un máximo de 20 nodos hoja. Tras el entrenamiento, se ha observado que el árbol generado contiene un total de 39 nodos y una profundidad total de 10. Con estos parámetros se ha obtenido una tasa de acierto del 0'92 en test.
3. **KNN:** Para este clasificador se han definido un total de 10 vecinos para tener en cuenta. Con esta configuración, el clasificador ha obtenido una tasa de acierto en test de 0'85.

Comparando los resultados de estas clasificaciones con los del dataset anterior, se puede observar que son menos acertados. Estos resultados indican que, clasificar únicamente las características estructurales y la posición de los nodos en la red (Patrón 2), no es suficiente para detectar con exactitud el tipo de red a la que pertenece cada usuario.

5.3.3 Dataset 3: Combinación.

Para el entrenamiento y clasificación de este dataset, se han establecido exactamente los mismos parámetros que para el dataset anterior en todos los clasificadores y se han obtenido resultados bastante mejores:

1. **Random Forest:** En esta ejecución, los árboles generados han tenido 35 nodos de media, y su profundidad máxima ha sido de 8. Con los mismos parámetros que anteriormente ha obtenido un porcentaje de acierto del 99% en los datos de test, lo que significa que el haber introducido la susceptibilidad al dataset ha hecho que la clasificación de los datos con este modelo sea un 5% mejor.
2. **Árbol de decisión**: Al ejecutar este modelo con los mismos parámetros para este dataset, se han generado un árbol con un total de 19 nodos y una profundidad de 7. El porcentaje de acierto obtenido con los datos de test ha sido del 98%, lo que implica un incremento del 6% en el número de aciertos del clasificador con respecto al dataset anterior.
3. **KNN:** En esta ejecución, el porcentaje de acierto obtenido de la clasificación de los datos de test ha sido del 99%, lo que implica una mejora del 11% con respecto al dataset anterior.

Una vez realizado el entrenamiento, se puede comprobar cuál de todos los atributos del dataset es el más importante a la hora de clasificar a los usuarios. A partir de los resultados de los tres datasets, se puede intuir que la susceptibilidad tiene un peso importante a la hora de decidir la clasificación del usuario, puesto que los modelos que se han entrenado con dicho atributo han obtenido mejores resultados que los que no lo han hecho. Mediante el atributo *feature_importance_* que el clasificador *Random Fore*st haya generado en la clasificación de este *dataset*, se podrá confirmar si la susceptibilidad es la característica más importante.

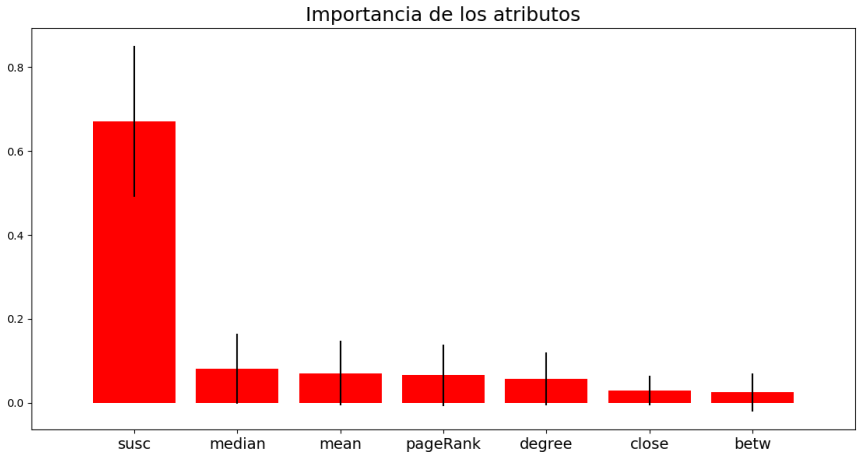


Figura 20: Importancia de los atributos en clasificación según RF

Podemos ver que, según la gráfica, *Random Forest* asigna al atributo se susceptibilidad, perteneciente al Patrón 1 una importancia mucho mayor que al resto de atributos.

5.4 Graph embedding: Aplicación y resultados

La aplicación de *Graph Embedding* puede tener dos finalidades distintas*:*

1. Por un lado, conocer el camino que sigue una noticia falsa y además detectar a los usuarios que se encargan de difundirla. El procedimiento sería el siguiente:
    a. Generar *random walks* sobre todo el grafo de la base de datos creada y calcular los N nodos más similares al autor de la noticia que se quiera inspeccionar.
    b. Una vez obtenidas las similitudes mediante la aplicación de Node2Vec, se puede profundizar un segundo nivel y obtener los usuarios más similares a los usuarios devueltos tras la primera iteración. Este proceso se repite tantas veces como sea necesario, hasta que se encuentre una posible ruta por la que presuntamente circule la noticia o se genere una subred de nodos conectados entre sí que hayan tenido relación con dicha noticia y por tanto hayan sido distribuidores de esta.
    c. De esta forma se consigue identificar a los usuarios que se encargan de distribuirla y además se puede conocer la posible ruta que seguirán las noticias que estén relacionadas con el grupo de usuarios obtenido.

2. Por otro lado, se puede utilizar para ubicar a un usuario en concreto dentro de la red *fake* o la *no fake* en función de los usuarios que obtengan una mayor similitud con el usuario inspeccionado. Un posible ejemplo para ilustrar este procedimiento sería el siguiente:
    a. Suponer que se tiene un nuevo usuario. Se procede a calcular todas las características de centralidad y la susceptibilidad que ya se han calculado para los demás usuarios de la red.
    b. Una vez calculadas dichas características, se procede a obtener listas de rutas aleatorias del grafo actual, que contiene al nuevo usuario y todas sus relaciones.

c. Se calcula entonces la similitud del nuevo usuario introducido con todos los usuarios de la red y se obtienen los 5 primeros, que se corresponden con los 5 usuarios más similares.
d. A partir de las relaciones que dicho usuario mantiene con el resto de los usuarios de la red, que ya están previamente etiquetados en función de los tweets con los que se relacionan, se realiza un conteo de los usuarios según pertenezcan al subgrafo de noticias *fakes* o *no fakes*. Dicho usuario será clasificado en función del número mayor que se haya obtenido a partir del conteo de los usuarios más similares a él

Se puede apreciar que el segundo proceso realizado es similar a aplicar KNN sobre todos los usuarios de la red, sin embargo, la principal diferencia es que las similitudes calculadas entre usuarios no se han hallado en base a sus características, sino a las relaciones que mantienen entre ellos escogidas de forma aleatoria.

6 Conclusiones y trabajo futuro

6.1 Conclusiones

A partir de la información obtenida de los usuarios de Twitter que se han considerado apropiados para este estudio. Se ha construido una base de datos en Neo4j sobre la que se han aplicado algoritmos sobre grafos, consiguiendo clasificar a los usuarios propensos a la propagación de noticias falsas de dos formas diferentes.

1. Por un lado, se han observado diferencias estructurales en función de la red que se esté examinando (*fake* o no *fake*). Mediante la generación de *datasets*, obtenidos a partir de agrupaciones de las características calculadas, se han aplicado diversos algoritmos de aprendizaje automático que han conseguido detectar y clasificar a los usuarios en función de la red a la que pueden pertenecer, obteniendo una tasa de acierto superior al 99% con determinadas combinaciones de datos.
2. Por otro lado, mediante la técnica de *Graph Embedding* se ha conseguido clasificar a los usuarios, en función de las similitudes calculadas mediante la aplicación del algoritmo Node2Vec sobre listas de *Random Walks*.

A partir de estas dos formas de clasificación de usuarios, se puede proceder a la detección de noticias falsas prestando atención a los tweets con los que se relacionan dichos usuarios. De esta forma, si se ha detectado que un usuario es asiduo en la difusión de *fake news*, se le puede prestar especial atención y examinar los tweets que mantengan cualquier tipo relación con él, y clasificarlos de igual forma como posible noticia *fake*.

6.2 Trabajo futuro

La detección de *fake news* es un área de investigación de la más absoluta actualidad. Si bien es cierto que se están empezando a crear e incorporar herramientas para su detección en redes sociales, aún queda mucho camino por recorrer. Por ese motivo, de cara a continuar este estudio se pueden definir algunos puntos interesantes para ampliar y mejorar la herramienta.

- En un futuro se podría aplicar la técnica de *Graph Embedding* para realizar predicciones de las uniones entre usuarios y así conocer por dónde se difundirá la noticia antes de que llegue a hacerlo.
- Si se logra conocer la posible ruta de la noticia antes de que esta ocurra, se podría intentar contener la noticia y así mitigar su impacto.
- La idea original, y un trabajo que queda pendiente por indicación del tutor, es integrar este trabajo junto con otro TFG ya realizado. Dicho TFG se encarga de aportar un valor de fiabilidad a cada noticia que examina. Mediante la unión de estos dos TFGs se puede construir un detector de *fake news* ya que, partiendo de un tweet calificado con un grado de fiabilidad se puede construir, de la forma en la que se ha explicado en el Capítulo 3, la subred de usuarios que estén relacionados con dicho tweet y, a partir de las características de los usuarios que se obtengan, aprender a identificar a los usuarios y ubicarlos en la red a la que pertenezcan. Esta integración excluirá la parte de la recolecta de datos que se ha llevado a cabo en este estudio y, en su lugar, utilizará los tweets clasificados mediante el TFG mencionado.

Referencias

[1] https://bigdatamagazine.es/cisco-preve-mas-trafico-ip-en-los-proximos-cinco-anos-que-en-toda-la-historia-de-internet consultado el 22 de junio de 2020

[2] https://es.wikipedia.org/wiki/Fake_news consultado el 22 de junio de 2020

[3] https://verne.elpais.com/verne/2016/11/16/articulo/1479308638_931299.html consultado el 22 de junio de 2020

[4] https://maldita.es/malditobulo/ consultado el 23 de junio de 2020

[5] https://www.politifact.com/ consultado el 23 de junio de 2020

[6] Gilda, S. (2017, December). Evaluating machine learning algorithms for fake news detection. In *2017 IEEE 15th Student Conference on Research and Development (SCOReD)* (pp. 110-115). IEEE.

[7] https://towardsdatascience.com/how-to-build-a-recurrent-neural-network-to-detect-fake-news-35953c19cf0b consultado el 12 de mayo de 2020

[8] Wang, W. Y. (2017). " liar, liar pants on fire": A new benchmark dataset for fake news detection. *arXiv preprint arXiv:1705.00648*.

[9] Shu, K., Bernard, H. R., & Liu, H. (2019). Studying fake news via network analysis: detection and mitigation. In *Emerging Research Challenges and Opportunities in Computational Social Network Analysis and Mining* (pp. 43-65). Springer, Cham.

[10] Tschiatschek, S., Singla, A., Gomez Rodriguez, M., Merchant, A., & Krause, A. (2018, April). Fake news detection in social networks via crowd signals. In *Companion Proceedings of the The Web Conference 2018* (pp. 517-524).

[11] https://www.statista.com/statistics/278414/number-of-worldwide-social-network-users/ consultado el 22 de junio de 2020

[12] https://es.wikipedia.org/wiki/An%C3%A1lisis_de_redes_sociales consultado el 24 de junio de 2020

[13] Diapositivas de la asignatura de Búsqueda y Minería de Información de la Universidad Autónoma de Madrid, curso 2019-2020, Pablo Castells. https://moodle.uam.es/course/view.php?id=71598

[14] Liu, B. (2011). Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data / (2nd ed. 2011. ed., Data-Centric Systems and Applications). Heidelberg: Springer.

[15] Easley, D., & Kleinberg, J. (2010). Networks, crowds, and markets: Reasoning about a highly connected world. New York: Cambridge University Press.

[16] https://es.wikipedia.org/wiki/Modelaje_matem%C3%A1tico_de_epidemias consultado el 25 de junio de 2020

[17] https://neo4j.com/top-ten-reasons/ consultado el 30 de mayo de 2020

[18] https://neo4j.com/docs/graph-data-science/current/ consultado el 30 de mayo de 2020

[19] Needham, M., & Hodler, A. E. (2019). *Graph Algorithms: Practical Examples in Apache Spark and Neo4j*. O'Reilly Media.

[20] https://medium.com/greyatom/decision-trees-a-simple-way-to-visualize-a-decision-dc506a403aeb consultado el 15 de junio de 2020

[21] https://towardsdatascience.com/understanding-random-forest-58381e0602d2#:~:text=The%20random%20forest%20is%20a,that%20of%20any%20individual%20tree. consultado el 15 de junio de 2020

[22] https://www.aprendemachinelearning.com/clasificar-con-k-nearest-neighbor-ejemplo-en-python/ consultado el 15 de junio de 2020

[23] https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTreeClassifier consultado el 17 de junio de 2020

[24] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html#sklearn.ensemble.RandomForestClassifier consultado el 17 de junio de 2020

[25] https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html#sklearn.neighbors.KNeighborsClassifier consultado el 17 de junio de 2020

[26] https://towardsdatascience.com/graph-embeddings-the-summary-cc6075aba007 consultado el 18 de junio de 2020

[27] Compagnon, P., & Olliver, K. (2016). *Graph embeddings for social network analysis, state of the art* (Doctoral dissertation, Master's thesis, INSA Lyon, 2016 (cit. on p. 22)).

[28] https://towardsdatascience.com/node-embeddings-node2vec-with-neo4j-5152d3472d8e consultado el 18 de junio de 2020

[29] https://snap.stanford.edu/node2vec/ consultado el 18 de junio de 2020

[30] https://neo4j.com/docs/graph-data-science/current/alpha-algorithms/random-walk/ consultado el 18 de junio de 2020

[31] http://docs.tweepy.org/en/latest/index.html consultado el 25 de mayo de 2020

[32] Zhou, X., & Zafarani, R. (2019). Network-based fake news detection: A pattern-driven approach. *ACM SIGKDD Explorations Newsletter*, *21*(2), 48-60.

[33] https://radimrehurek.com/gensim/index.html consultado el 20 de junio de 2020

[34] https://neo4j.com/docs/graph-data-science/current/alpha-algorithms/random-walk/ consultado el 20 de junio de 2020

Glosario

API	Application Programming Interface
SNA	Social Network Analysis

Anexos

A Datasets utilizados

1. Dataset 1: Susceptibilidad.

```
      susc  label
0    1.000      1
1    1.000      1
2    0.375      1
3    1.000      1
4    1.000      1
..     ...    ...
951  0.000      0
952  0.000      0
953  0.000      0
954  0.000      0
955  0.000      0
[956 rows x 2 columns]
```

2. Dataset 2: Centralidad.

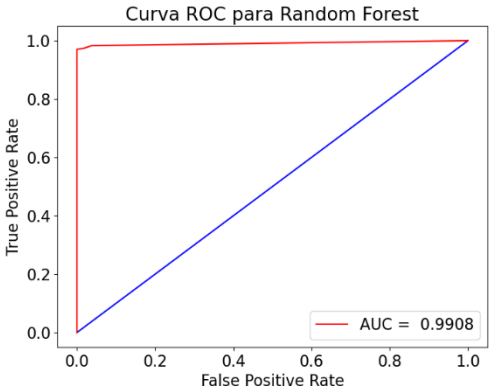
```
              mean      median         betw     close  degree  pageRank  label
0         1.288118    0.076235     0.000000  0.000000       5  0.152470      1
1         1.038687    0.077373     0.000000  0.000000       4  0.154746      1
2         6.315642    2.053990     1.614507  2.493473      21  0.154589      1
3      1329.150494  147.243490  5021.960409  2.486979     292  0.154589      1
4         1.324036    0.146749     0.138095  0.002646       5  0.155403      1
..             ...         ...          ...       ...     ...       ...    ...
951      13.801289    0.102579     0.000000  0.000000      55  0.205158      0
952       4.539139    0.078279     0.000000  0.000000      18  0.156558      0
953       4.045271    0.090542     0.000000  0.000000      16  0.181084      0
954      13.801289    0.102579     0.000000  0.000000      55  0.205158      0
955      13.801289    0.102579     0.000000  0.000000      55  0.205158      0
[956 rows x 7 columns]
```

3. Dataset 3: Combinación.

```
              mean      median         betw     close  degree  pageRank   susc  label
0         1.288118    0.076235     0.000000  0.000000       5  0.152470  1.000      1
1         1.038687    0.077373     0.000000  0.000000       4  0.154746  1.000      1
2         6.315642    2.053990     1.614507  2.493473      21  0.154589  0.375      1
3      1329.150494  147.243490  5021.960409  2.486979     292  0.154589  1.000      1
4         1.324036    0.146749     0.138095  0.002646       5  0.155403  1.000      1
..             ...         ...          ...       ...     ...       ...    ...    ...
951      13.801289    0.102579     0.000000  0.000000      55  0.205158  0.000      0
952       4.539139    0.078279     0.000000  0.000000      18  0.156558  0.000      0
953       4.045271    0.090542     0.000000  0.000000      16  0.181084  0.000      0
954      13.801289    0.102579     0.000000  0.000000      55  0.205158  0.000      0
955      13.801289    0.102579     0.000000  0.000000      55  0.205158  0.000      0
[956 rows x 8 columns]
```

B Curvas ROC y matrices de confusión

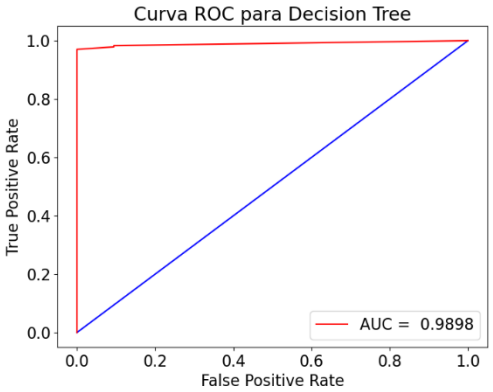
1. Dataset 1: Susceptibilidad.



Matriz de confusión:

5	1
7	227

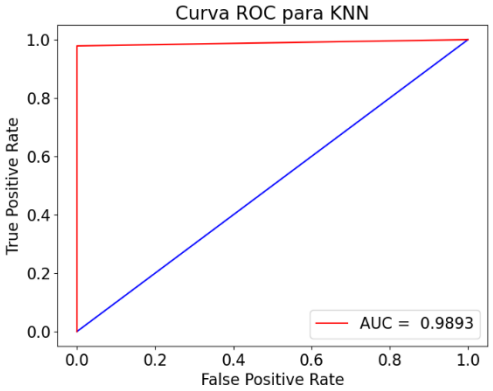
Figura 21: Curva ROC resultante de la ejecución de RF sobre el Dataset 1



Matriz de confusión:

53	0
3	231

Figura 22: Curva ROC resultante de la ejecución de Árbol de Decisión sobre el Dataset 1

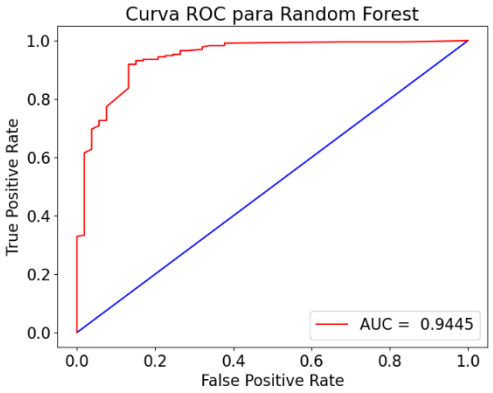


Matriz de confusión:

53	0
5	229

Figura 23: Curva ROC resultante de la ejecución de KNN sobre el Dataset 1

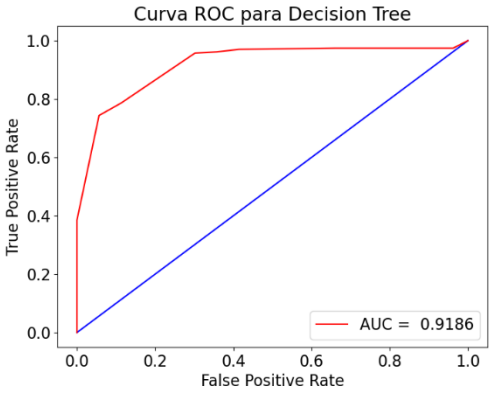
2. Dataset 2: Centralidad



Matriz de confusión:

40	13
11	223

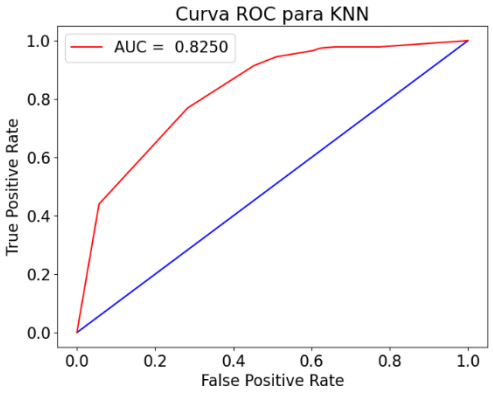
Figura 24: Curva ROC resultante de la ejecución de RF sobre el Dataset 2



Matriz de confusión:

36	16
10	224

Figura 25: Curva ROC resultante de la ejecución de Árbol de Decisión sobre el Dataset 2

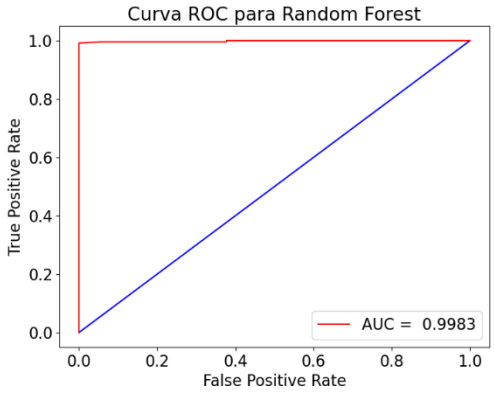


Matriz de confusión:

26	27
13	221

Figura 26: Curva ROC resultante de la ejecución de KNN sobre el Dataset 2

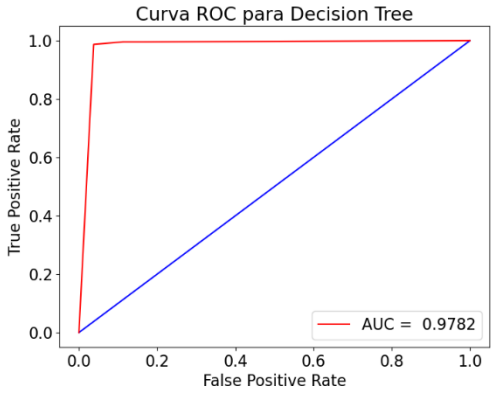
3. Dataset 3: Combinación



Matriz de confusión:

53	0
2	232

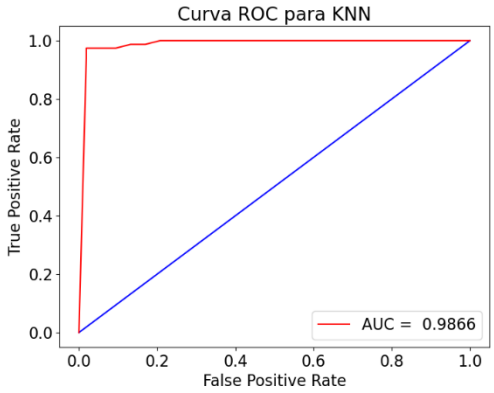
Figura 27: Curva ROC resultante de la ejecución de RF sobre el Dataset 3



Matriz de confusión:

51	2
3	231

Figura 28: Curva ROC resultante de la ejecución de Árbol de Decisión sobre el Dataset 3



Matriz de confusión:

48	5
6	228

Figura 29: Curva ROC resultante de la ejecución de KNN sobre el Dataset 3